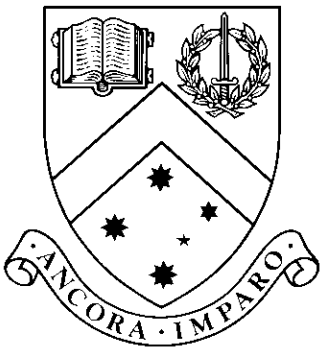
Light-Weight and Adaptive Reasoning for Mobile Web Services

by

Luke Albert Steller, BNetComp(Honours)



Thesis

Submitted by Luke Albert Steller

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. Shonali Krishnaswamy

Associate Supervisor: Dr. Mohamed Medhat Gaber

Caulfield School of Information Technology
Monash University

August, 2010

Contents

List of Tables

List of Figures

Light-Weight and Adaptive Reasoning for Mobile Web Services

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Luke Albert Steller
August 21, 2010

Notice 1

Under the Copyright Act 1968, this thesis must be used only under the normal conditions of scholarly fair dealing. In particular no results or conclusions should be extracted from it, nor should it be copied or closely paraphrased in whole or in part without the written consent of the author. Proper written acknowledgement should be made for any assistance obtained from this thesis.

Notice 2

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Acknowledgments

The process involved to complete this PhD was an extremely difficult, at times stressful and on-going process which never seemed to end. Therefore, the completion of this dissertation and all the work which has gone into it, is extremely gratifying. There are a few people who deserve thanks for their extremely important contribution to this process.

I would like to thank Dr. Shonali Krishnaswamy for her willingness to take on the responsibilities associated with supervising me from beginning to end of my candidature. Shonali has played the primary role in assisting me in my research by investing significant time and effort into my work. Shonali's input of ideas, enthusiastic encouragement and support during regular meetings has been invaluable to the completion of this thesis. Her eye for detail particularly in reading every word in my dissertation many times over has also been extremely valued. It most likely would not have been possible to bring this thesis to completion without Shonali's involvement.

Dr. Mohamed Gaber also deserves thanks for being willing to supervise me during the latter part of my candidature. He provided extremely valuable discussion and suggestions regarding my research and the writing of this thesis. Thanks to Dr. Seng Loke for being kind enough to offer input and suggestions to my research.

I also express acknowledgement to my parents who always supported and encouraged a strong belief in the high value of education and provided financial support in order to complete the PhD. Many thanks to Paul Blaich for helping me proof read the thesis and for his general support.

There are many people that I have met as a result of doing a PhD. I am grateful to have met my desk neighbour Eddie Leung, and Michael Nebeling as a result of a memorable conference. Thank you to all my research colleagues in DSSE who have always been extremely friendly and helpful, including Karen Mitra, Saguna, Pari Delir Haghighi, Kutilia Gunasekera, Prem Jayaraman, Sunam Pradhan and any other DSSE students who I have not named. I would like to thank the staff at the Caulfield School of IT for their friendly and helpful support. Thank you also to Dr. Angela Carbone, Dr. Chris Ling, Dr. Phu Dung Le and Janet Fraser for providing me with teaching opportunities.

I begun my candidate at the Peninsula School Network Computing and I would like to acknowledge the staff and fellow research students who I worked with during that time. Thanks also goes to Dr. Jan Newmarch for his supervision and input to my research during the early stages of my PhD. I am also glad to have had the pleasure of working with those fellow research students from Struan, such as Nisha Leena Sehna Roy, Jackie How Keat Low, Paulo Tam, Chu Tiong Yeoh, Robert Bram, Craig McDonald and Adrian Ryan.

Luke Albert Steller

Monash University

August 2010

Outcomes

The outcomes in this dissertation have been reported in the following publications:

Referred Journal:

- Steller, L., Krishnaswamy, S., and Gaber, M.M., Enabling Scalable Semantic Reasoning for Mobile Services, In *International Journal of Semantic Web Information Systems - Special Issue on Scalability and Performance of Semantic Systems*, Vassilis Christophides, Stefan Decker and Jeff Heflin (Eds.), IGI **5**(2): p. 91 - 116

Referred Conference Proceedings:

- Steller, A. L, Krishnaswamy, S., and Gaber, M.M. A Weighted Approach to Partial Matching for Mobile Reasoning, In Proc. *International Semantic Web Confidence (ISWC '09)*, Springer-Verlag, Virginia, USA, pp. 618 - 633;

- Steller, L., Krishnaswamy, S., and Gaber, M.M., Cost Efficient, Adaptive Reasoning Strategies for Pervasive Service Discovery, In Proc. *International Conference on Pervasive Services (ICPS '09)*, ACM, London, UK, pp. 11 - 20;

- Steller, L., Krishnaswamy, S. Efficient Mobile Reasoning for Pervasive Discovery, In Proc. *Symposium on Applied Computing (SAC '09), Session on Semantic Web and Applications*, ACM, Honolulu, Hawaii, pp. 1247 - 1251;

- Steller, L., and Krishnaswamy, S., (2008), Optimised Semantic Reasoning for Pervasive Service Discovery, In Proc. *International Conference on Service Oriented Computing (ICSOC '08)*, Springer-Verlag, Sydney, Australia, pp. 620 - 625;

- Steller, L., and Krishnaswamy, S., (2008), Optimised Mobile Reasoning for Pervasive Service Discovery, In Proc. *International Conference on Web Services (ICWS '08)*, IEEE, Beijing, China, pp. 789 - 790;

- Steller, L., and Krishnaswamy, S., (2008), Pervasive Service Discovery: mTableaux Mobile Reasoning, In Proc. *International Conference on Semantic Systems (I-SEMANTICS '08)*, Graz, Austria, pp. 93 - 101;

- Steller, L., Krishnaswamy, S., Cuce, S., and Newmarch, J., (2008), A Weighted Approach to Optimised Reasoning For Pervasive Service Discovery Using Semantics and Context, In Proc. *10th International Conference on Enterprise Information Systems (ICEIS '08)*, Barcelona, Spain, pp. 113 - 118.

Referred Workshop Proceedings:

- Steller L., Krishnaswamy S. and Newmarch J., Discovering Relevant Services in Pervasive Environments Using Semantics and Context, In Proc. *3rd International Workshop on Ubiquitous Computing (IWUC-2006) in Conjunction with the International Conference on Enterprise Information Systems (ICEIS '06)*, Paphos, Cyprus, pp. 3 - 13.

Light-Weight and Adaptive Reasoning for Mobile Web Services

Luke Albert Steller, BNetComp(Honours)
Monash University, 2010

Supervisor: Dr. Shonali Krishnaswamy
Associate Supervisor: Dr. Mohamed Medhat Gaber

Abstract

The growth of smart phones and PDAs coupled with the emergence of Web Services as the de facto technology for supporting seamless heterogeneous integration has led to an emerging focus on mobile services. This emergence of mobile services necessitates service selection mechanisms that are accurate and efficient. Service selection involves matching of a user's requirements against the available services in the user's environment. It is well established that accuracy in service matching is improved by the use of semantics as opposed to simpler approaches such as keyword / interface matching. Semantic matching is performed by semantic reasoners. Current approaches to semantics based service selection tend to perform matching using external / remote high performance servers, because reasoning is a computationally complex and resource intensive activity that does not scale well to mobile devices.

However, there are several advantages of performing semantic service matching on-board the mobile device. For instance, on-board matching avoids the overheads associated with the provision and maintenance of external servers to perform matching remotely. Additionally, continuous network access to a remote server has been shown to be a relatively higher drain on a mobile device's battery power when compared to processing activities. Furthermore, a connection may not always be available since mobile devices suffer from intermittent connectivity and frequent disconnection. There may also privacy

concerns with transmitting sensitive data to a third party remote server (e.g. a user's shopping preferences and habits).

Therefore, in this thesis we propose and develop a novel light-weight and adaptive approach for on-board semantic mobile matching. This thesis makes two significant contributions. Firstly, due to computational complexity, current reasoners cannot perform matching of large ontologies on mobile resource constrained devices. Therefore, we propose and develop mTableaux which enables mobile semantic matching by performing optimisations of the well-known Tableaux algorithm that is used in many of the state-of-the-art open source and commercial reasoners today. These optimisations result in improving the computational efficiency of the semantic reasoning process with a specific focus on scaling to mobile devices, without significantly reducing result accuracy. Secondly, current reasoners typically produce only a positive or negative result under an "all or nothing" principle in which the matching task must be completed in full before a result is provided. Therefore, we propose and develop an adaptive and incremental approach to deliver the outputs of a reasoning task. This allows a mobile user to get valid partial results from a reasoner depending on constraints such as changing context, time or availability of computational resources.

We have implemented our proposed light-weight and adaptive reasoning strategies, and conducted extensive experimental performance evaluations which clearly demonstrate that our strategies improve response time and enable incremental matching. Our performance evaluations clearly demonstrate that the efficiency improvements in response time do not compromise accuracy. This evaluation includes tests on a resource constrained mobile device and a comparison of our approach against commercial and open source reasoners in desktop environments, using two realistic application scenarios as well as publicly available ontologies.

In summary this dissertation has addressed the problem of enabling efficient and accurate mobile reasoning on small devices to meet dynamic resource levels and user needs in mobile environments. The research done over the course of this dissertation has been published in one international journal paper, seven conference papers and one workshop paper.

Chapter 1

Introduction

1.1 Preamble

Today, a range of mobile devices are penetrating the market, such as hand held Personal Computers (PCs)[1], Personal Digital Assistants (PDAs)[2], smart phones[3], devices in vehicles[4] and other wearable devices (de Freitas and Levene, 2003; Billinghurst and Starner, 1999). More recently, the sale of smart phones has exploded (Gartner, 2008). Thus, there are exciting potential opportunities to develop innovative mobile applications to meet specific user needs. The power of a smart mobile device lies in its interaction capability (Weiser, 1991). That is, its ability to communicate, share information and interact with the user's environment in a way which is meaningful to the user, thus enabling "communication on the move" (Zabariadis and Doshi, 2004). These software applications on mobile devices must seamlessly adapt and interact with the surrounding environment in keeping with the pervasive computing vision (Ferscha, 2009; Satyanarayanan, 2001).

Another key technology that has emerged in recent years is Web Services, which is defined as "a software system designed to support interoperable

[1]http://www.silicon.com/technology/mobile/2006/02/13/analysis-what-is-a-smart-phone-39156391/ (accessed May 2009)
[2]http://reviews.cnet.com/pdas/ (accessed May 2009)
[3]http://www.cnet.com/smartphones/ (accessed May 2009)
[4]http://reviews.cnet.com/gps/ (accessed May 2009)

machine-to-machine interaction over a network"[5]. Web Services are the natural evolution of the World Wide Web, from a source of information to an open medium for facilitating complex software interactions across the Web using open standards. This transformation provides an open distributed environment for dynamic interactions between different / heterogeneous software components.

The integration of the service oriented paradigm with mobile / pervasive computing is leading to increasing research focus into mobile services. A key focus of mobile services is on discovering and accessing external services from mobile devices, leveraging the communication capabilities of these devices. A key challenge is to identify services that are relevant to the user's changing context (El-Sayed and Black, 2006), such as location (Dietze et al., 2009; Doulkeridis and Vazirgiannis, 2008), and device connectivity levels / QoS (Niazi and Mahmoud, 2009; Dietze et al., 2009; Preuveneers and Berbers, 2008a; Mokhtar et al., 2008; Vu et al., 2007, 2005), etc. A second emerging focus in mobile / pervasive services is to leverage the increasing computational capabilities of today's mobile devices to host services (Aijaz et al., 2009; Schmidt et al., 2008; Tergujeff et al., 2007; Srirama et al., 2006; Asif et al., 2007) that may be accessed by both the user and other devices typically in a localised area.

This thesis focuses on developing infrastructure to support pervasive services to be deployed on mobile devices. We take an important step in enabling pervasive services on mobile devices to be enhanced with semantic reasoning capabilities to support more sophisticated matching of user requests with service capabilities (Peng et al., 2008; El-Sayed and Black, 2006).

[5]http://www.w3.org/TR/ws-arch/ (accessed May 2009)

1.2 Motivations

In this section, we motivate the need for on-board semantic reasoning on mobile devices. As stated in the previous section, Web Services provides an open standard for interacting with heterogeneous software applications and information sharing over the Internet. In the context of mobile or pervasive services there are two primary modes of operation:

1. static services which are hosted on centralised / remote high-end servers;
2. mobile services which are hosted on the devices themselves.

However, while Web Services enables a client user or application to interact with a provider application, a mechanism is required to select a service from potentially many services available, which best meets the requirements of the user (Trastour et al., 2001). For instance, consider the following situations which would require a mechanism to match user requirements with advertised service descriptions:

- **Discovery of services in a local precinct:** A mobile user has just arrived in Sydney airport and wishes to discover service descriptions about a WiFi Internet cafe;
- **"Infrastructure-less" peer-to-peer (P2P) information sharing:** A large group of students are on a field trip in a remote location where there is no infrastructure to provide fixed or mobile Internet access. Therefore, the students form an ad-hoc mobile network to facilitate collaboration of field results. A student wishes discover data which is hosted remotely on another student's device in his local area (Chatti et al., 2006);
- **On-device services management:** There is an abundance of services and applications which can be installed or removed from a mobile user's

own device on a needs basis. For example, Google[6] and Yahoo[7] offer many mobile applications such as blogging, news, finance and sports. For instance, the Apple iPhone[8] has thousands of "apps. for everything"[9]. With more and more mobile applications being published as services, the user needs a mechanism to help find the application which meets his or her particular requirements.

When a mobile device needs to search for a particular service, such as in the in the examples above, this presents additional challenges for service matching mechanisms. Small mobile devices are typically resource constrained in terms of processing power, memory capabilities, screen size, battery life, etc. Despite this, due to the dynamic nature of mobile environments, mobile users require an answer quickly (Roto and Oulasvirta, 2005), while maintaining a high level of result accuracy (Kargin and Basoglu, 2007).

Early mechanisms for service matching involved keyword and interface matching approaches such as the Universal Description and Discovery Integration (UDDI)[10], which was a centralised registry of Web Services. However, UDDI failed obtain widespread usage and support (Hartman and Reynolds, 2004; SysCon, 2005). A key problem with keyword based approaches is that they do not provide accurate results (Bernstein and Klein, 2002). Improved accuracy can be achieved by capturing semantics (Abramowicz et al., 2008). To overcome this problem, semantic languages (Martin et al., 2007) were developed to describe Web Services (Li and Horrocks, 2004; Hepp, 2006). These languages improve accuracy of matching (Bernstein and Klein, 2002), by using semantic reasoners (Gonzalez-Castillo et al., 2001).

Most current service matching approaches off-load the matching process to remote servers which perform this process on behalf of the mobile device

[6]http://www.google.com/mobile (accessed May 2009)
[7]http://www.yahoo.com/mobile (accessed May 2009)
[8]http://www.apple.com/iphone (accessed May 2009)
[9]http://www.apple.com/iphone/apps-for-everything (accessed May 2010)
[10]http://uddi.xml.org (accessed May 2009)

(Li et al., 2001), such as Doulkeridis and Vazirgiannis (2008); Baousis et al. (2008); de Andrade et al. (2007); Chen et al. (2006); and Suraci et al. (2007). However, this thesis aims to support matching of user requests with service descriptions using semantic reasoners, on-board the mobile device. In order to motivate this aim, we will now revisit each of the scenarios from earlier in this section:

- **Discovery of services in a local precinct:** Currently, Sydney airport provides touch screen kiosk terminals which provide information about retail outlets and services provided in the airport, as illustrated in figure 1.1. This allows a user to locate an Internet cafe as described earlier in this section. Touch screens suffer from poor scalability (only one person can use them at a time), a fixed location (the user must walk to a touch screen) and support only very simple searches. With the increasing abundance of PDAs and smart phones, we propose these kiosks offer semantic service descriptions for download by small devices (either using short range communications technologies as part of a bluetooth or a WiFi connection). For example, when a mobile user walks past a kiosk, his or her mobile device can download service descriptions and the user can then perform sophisticated semantic matching when required, as the user walks around the airport. This approach is a natural extension of the current kiosk model, in which a kiosk is simply deployed and does not require the significant cost or considerations associated with centralised server provision for semantic discovery requests which has additional overheads of reliability, security and maintenance;

- **"Infrastructure-less" P2P information sharing:** Semantic matching would enable more accurate matching of user requests for specific field results to those results which have been collected by other students. This semantic matching process would need to be completed on-board the mobile user's device since there is no high-end server available in a

remote environment, nor is it physically or financially feasible to provide one;

- **On-device services management:** Semantic descriptions would allow mobile users to install applications which more accurately suit their requirements. Since the decision is about applications on-board the device itself, matching too, should occur on-board.

Figure 1.1: Sydney airport store finder kiosk

Each of the scenarios above, motivate the need to support on-board semantic reasoning, rather than providing a centralised high-end server because this provision is either not possible / feasible or has significant cost drawbacks. We summarise the main drawbacks of a server based approach for these scenarios as follows:

- **Availability:** in some environments it is not physically possible nor feasible to provide centralised server infrastructure and associated network infrastructure (Srirama et al., 2007). In these cases communication takes place via temporary Peer to Peer (P2P) networks between mobile nodes (Gehlen and Pham, 2005; Choi et al., 2005);

- **Battery Usage:** continuous network connectivity involved in a remote server approach results in a high cost to battery usage. In fact, network communication requires more battery usage than CPU processing

(Reaghunathan et al., 2002). Additionally, in keeping with Moore's law (Moore, 2006, p. 90 - 114), technological advances in hardware processing capability continues to outstrip the progress in battery capacity;

- **Disconnection:** mobile devices suffer from intermittent connectivity and frequent disconnection, especially when mobile users are on the move (they may move out of network range) (Ott and Kutscher., 2004). Matching may be delegated to a higher end static node in the network which itself may be inaccessible due to connectivity issues. Reliance on a continuous connection to a high-end server means that the matching task is prone to failure when the network connection or service to which the matching is delegated, becomes unavailable;

- **Cost:** significant costs may arise from providing centralised infrastructure to perform service matching. In particular, a centralised scheme would need to be reliable, secure and would require maintenance. Reliability considerations involve providing wireless network connectivity over a large area and capacity to handle rapid increases in the number of mobile user's attempting to perform matching at the same time. Providing the necessary infrastructure / redundancy to avoid performance delays or server failures (Wang and Hu, 2008) during peak times, may result in significant costs to the end user. Cost is a significant factor which influences whether a user is likely to utilise a mobile service and studies have clearly shown that the benefit must exceed the costs involved (Kargin and Basoglu, 2007);

- **Privacy:** improved accuracy in service discovery and matching can be achieved by utilising contextual data relating to the user's environment. This contextual data may relate to the user's current situation, or the user's previous requests and habits. However, current and historical context data can in some cases be extremely sensitive. Transmitting this

information to a third party server and reasoning on this data could raise privacy concerns for the mobile user, in terms of whether the server itself and the mode of transmission is trustworthy (Kleemann, 2006).

Employing a decentralised approach in which the matching occurs on the mobile device itself overcomes the drawbacks listed above. On-board reasoning alleviates the availability and disconnection problems because no network connection is required once the service descriptions have been downloaded onto the mobile device. Service descriptions can be exchanged by mobile node peers, provided by a short-range download point (such as a kiosk or shopping centre entrance) or downloaded previously from the Internet (e.g. at home or work). Descriptions can be stored on removable secondary storage media such as an Secure Digital (SD) card which is inexpensive and can store several gigabytes of data. Battery usage is minimised since CPU processing utilises less energy than network transmission (Reaghunathan et al., 2002). Costs are minimised if no server or network provision is required, apart from temporary short-range network transmission for service description download. Any number of devices can be deployed without requiring additional provision of infrastructure to support increases in capacity for scalability and the costs associated with this. Privacy concerns are alleviated since all processing occurs on-board the device. Sensitive data and conclusions drawn from this data are never transmitted to an external party.

However, although on-board semantic reasoning provides improved accuracy and overcomes the aforementioned drawbacks, it has not been adopted because current semantic reasoners scale poorly (Zacharias et al., 2007) and are too resource intensive to be deployed on resource constrained mobile devices. For instance, Zoric et al. (2007b) found that matching with the assistance of a reasoner is 3-4 times more time consuming than without a reasoner even in desktop environments. As such, logic reasoners which perform semantic matching cannot easily be ported to resource constrained devices. Figure 1.2

illustrates an out of memory error when reasoning is attempted using the Pellet[11] reasoner on a small device. Thus, the current state-of-the-art with respect to mobile service matching is to either use keyword based technologies which trade-off-accuracy to achieve higher performance or alternatively to have an external service infrastructure to manage semantically driven semantic matching.

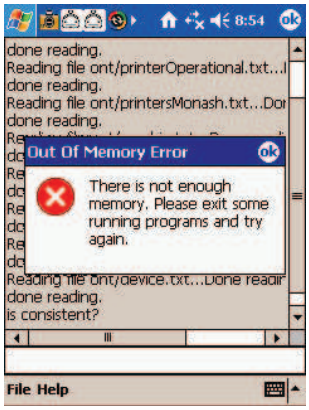


Figure 1.2: Not enough memory to perform semantic matching on a HP iPAQ PDA using standard Pellet reasoner

Current reasoners are resource intensive primarily because they adhere to strict notions of generating correct and complete results (Fensel and van Harmelen, 2007). However, mobile environments introduce a new set of priorities for information use. The need for correct information is important but it is equally necessary for information to be provided with a quick response time. Therefore, our approach is to relax these notions of completeness to obtain substantial performance improvements without significantly reducing accuracy. Our approach is two fold, firstly we propose and develop strategies which significantly improve the efficiency of mobile reasoning by relaxing the guarantee for completeness. In addition, we introduce flexibility by proposing and developing a resource-adaptive priority matching approach. This approach prioritises the user requirements based on importance to the user and incrementally matches the most important inference conditions first. The reasoner

[11] http://clarkparsia.com/pellet/ (accessed May 2009)

can then be stopped at any stage to obtain a partial result, without completing the matching task in full, as is required by current reasoners. In the next section we outline the specific objectives of this thesis, underpinned by the motivations described in this section.

1.3 Objectives and Contributions

We aim to provide accurate and fast on-board matching of mobile user requirements with mobile services. Improved matching accuracy can be achieved by utilising semantic languages. However, semantic reasoning is significantly resource intensive. Therefore, current reasoners cannot be ported to mobile resource constrained devices. Generally, current mobile systems offload processing to high-end servers (Li et al., 2001). However, continuous access to reliable network and server infrastructure is not possible in all situations and may come at a significant financial cost which may be passed on to the end user. These drawbacks can be overcome if matching occurs on-board the mobile device itself. We summarise the contributions of this thesis as follows:

1. The proposal and development of optimisation and caching strategies to enable and improve the efficiency of on-board mobile reasoning. The optimisation strategies focus on improving performance efficiency by relaxing the strict adherence to completeness. The caching strategy stores the results of previous requests so that they can be used in similar future requests. Given the growth of secondary storage media, we incorporate caching mechanisms to semantic matching tasks in order to improve response time;

2. The proposal and development of a resource adaptive priority reasoning strategy. This strategy supports incremental reasoning which can be interrupted prematurely if there are insufficient resources or time available

to continue matching. It also provides a weighted degree of match to the user, based on the matching completed;

3. The evaluation of the optimisation / caching and the adaptive inference strategies. We evaluate our optimisation and caching strategies by both comparing these with other widely used reasoners which do not run on a resource constrained mobile device and performing matching tasks on a resource constrained device using these strategies. We also evaluate our priority based, incremental adaptive inference strategy by performing various priority based matching tasks on a resource constrained device under certain time constraints. These evaluations demonstrate the significant performance improvements which our proposed approaches bring to mobile semantic reasoning, thus, enabling matching on a resource constrained device.

This research has resulted in (Steller et al., 2009b,a,c; Steller and Krishnaswamy, 2009, 2008b,a,c; Steller et al., 2008, 2006).

1.4 Thesis Structure

This thesis is structured into seven chapters. Chapter 2 contains a review of related research. In this chapter we provide a review of matching mechanisms including semantic languages and associated reasoners which provide greater accuracy for matching than simple approaches such as keyword matching. Since the main goal of our thesis is to facilitate accurate and efficient mobile matching, we provide a review of current approaches for service matching and categorise these based on whether they support semantic matching and perform matching on-board a mobile device. However, while semantic matching improves accuracy compared to keyword based approaches, it is a computationally complex and resource intensive task due to an emphasis on result completeness. Therefore, it does not scale well to mobile devices. As such,

we identify a need for a mobile semantic reasoning approach which trades accuracy with performance efficiency, to support matching of user requests with semantic service descriptions in realistic scenarios.

In Chapter 3 we provide a background of Description Logic and the Tableaux inference proof algorithm, which are widely used by current open source and commercial semantic reasoners. This background is required in order to understand the contributions of our dissertation which are presented in Chapters 4 and 5.

In Chapter 4, since current reasoners are too computationally complex to perform matching of large ontologies on resource constrained devices, we propose and develop optimisation and caching strategies for the Tableaux inference proof to enable mobile reasoning by improving response time without significantly reducing accuracy. We propose and develop two optimisation strategies which limit the size of the reasoning task based on the user request and the service description being checked in the matching process. We also propose and develop a strategy which caches the evaluations of the reasoner so that these can be used to improve response time of similar reasoning tasks.

Additionally, current reasoners provide only a match or failed match result, and require that the matching task be completed in full before any result is provided based on an "all or nothing" principle. Therefore, in Chapter 5, we propose and develop our resource adaptive inference strategy which takes into account important user constraints in a mobile environment such as time and resources. Our strategy incrementally matches conditions in the user request in priority order of importance to the user. The matching process can be interrupted based on user constraints such as available time and resources to provide a match result based on the matching completed.

In Chapter 6 we detail the prototype implementation of our proposed strategies from Chapters 4 and 5 and present extensive experimental validation for our contribution. We provide an evaluation of computational performance

efficiency and accuracy of our strategies against other currently available semantic reasoners in the desktop environments in which these reasoners operate in. We also provide an evaluation of response time efficiency of our strategies a mobile device. Our experiments demonstrate that our optimisation / caching strategies improve response times without compromising result accuracy and that our adaptive inference strategy effectively enables incremental reasoning.

In Chapter 7 we conclude the dissertation and summarise our key contributions and findings. We also outline future directions for this work.

Chapter 2

A Review of Service Matching for Mobile Environments

2.1 Introduction

There has been considerable growth and proliferation of mobile devices. For instance, world wide sales growth of smart phones was 29 percent in the first quarter of 2008 alone (Gartner, 2008). In Australia, laptop sales outstripped desktop PCs in 2008 and smart phones are expected to account for 57 percent of the market, which is up from 16 percent in the previous year (Buchanan, 2009). Veijalainen (2008) states that the number of mobile subscribers world wide is reaching the 3 billion mark, with the most significant growth being in developing countries such as India and China, where mobile devices are the only mode of access to the Internet. In India for instance, Sridhar (2007) shows that the growth of mobile subscribers has been exponential in the last 10 years.

This provides significant opportunities for new mobile applications which meet the need of mobile users to access to relevant information in their environment. There has already been substantial growth in the development of

new mobile applications. For instance, the apple iPhone[1] has thousands of "apps. for everything"[2].

The growth in the number of mobile devices, computational capability of these devices and in the availability of advanced networking and communication infrastructure lays the foundation for "pervasive computing" (Zabariadis and Doshi, 2004; Ferscha, 2009; Satyanarayanan, 2001). Central to the vision of pervasive computing is that the environment will be highly populated with networked devices such that these can be gracefully integrated with human users (Satyanarayanan, 2001). Environments may include the home, car, airport, office etc, populated with devices such as home appliances, PCs, smart phones, PDAs, projectors, GPS systems etc. Han et al. (2006) suggests that a majority of the workforce spend at least 20 percent of their time away from their desk and that the productivity of these workers can be improved by 30 percent when proper mobile technologies are deployed.

In order to support the interaction of heterogeneous applications in an environment with a high level of mobility, Xiaosu and Jian (2005) advocate the use of a Service Oriented Architecture (SOA) such as Web Services (WS)[3]. Web Services enable loosely coupled interoperability for machine-to-machine interaction over the World Wide Web using open standards. This provides the necessary open distributed environment that allows dynamic interactions between different / heterogeneous software components. In the context of leveraging the Web Services paradigm to support mobile users, there are principally two new modes of operation:

1. External / remotely hosted Web Services accessed by mobile clients;

2. Mobile hosted Web Services.

Advances in computational capabilities and networking provisions mean that mobile devices are increasingly capable of hosting their own Web Services

[1]http://www.apple.com/iphone (accessed May 2009)
[2]http://www.apple.com/iphone/apps-for-everything/ (accessed May 2010)
[3]http://www.w3.org/TR/ws-arch/ (accessed May 2009)

(Aijaz et al., 2009; Schmidt et al., 2008; Tergujeff et al., 2007; Srirama et al., 2006; Asif et al., 2007). In addition, services can be migrated to another device if a mobile node moves out of an available range (Kim and Lee, 2007; Preuveneers and Berbers, 2008b).

However, as we identified in the previous chapter, the problem remains in terms of how to match user requirements with services available. In addition, as described in Section 1.2, due to the challenges of infrastructure availability, the costs involved for its provision, the battery usage involved for network communication and privacy concerns about transmitting sensitive user data, a centralised server for matching is not always appropriate or applicable. Therefore, the main focus of this thesis is to support on-board matching of a user request with service descriptions using a mobile device. Due to their inherent dynamic environment, mobile users also have an increased need for relevant information and services to be accessed relatively quickly, which we discuss as follows:

1. **Accurate / relevant matching:** mobile users require useful and accurate matching of their needs against the available services (Kargin and Basoglu, 2007). In a dynamic mobile environment the matching process must take into consideration the information which is currently available in that environment to accurately match a service which is currently relevant to the user in terms of the user's current needs and situation. In addition, mobile devices often have a small display terminal, making it more difficult for a mobile user to choose from several match results, compared to a large PC screen (Peng et al., 2008). Accuracy and relevance is important because the user is likely to want to act immediately on match results;

2. **Efficient / fast matching:** the performance of the matching process is a key consideration in mobile environments. A mobile user will often require services related to a task that the user is currently undertaking

or about to undertake (i.e. the user will act on the results immediately). In addition, mobile environments are extremely dynamic, implying there are many factors and events which may easily draw the user's attention away from the mobile device. Roto and Oulasvirta (2005) suggests that mobile users generally have a tolerance of 5 to 15 seconds in response time.

Improved accuracy with respect to matching can be best achieved using semantic languages (Thanh and Jorstad, 2005) such as the Web Ontology Language (OWL)[4] to describe user requests and services. This enables the accurate selection of a service from potentially, many services available, depending on which service best suits the user's requirements (Trastour et al., 2001). However, the use of semantics requires the deployment of semantic reasoners which are considerably resource intensive (Zacharias et al., 2007). For instance, Zoric et al. (2007b) found that matching with the assistance of a reasoner is 3-4 times more time consuming than without a reasoner in desktop environments. This has prevented the use of semantic processing on small resource constrained devices. Therefore, the main focus of this thesis is to propose and develop strategies for light-weight and resource-adaptive semantic inference, such that the requirements of performance / response time needs of mobile services users, is underpinned by accurate matching.

This chapter presents a review of service matching approaches for mobile environments. The chapter is organised as follows: Section 2.2 outlines mechanisms which are used for matching requests with service descriptions ranging from keyword / interface matching to semantic based matching. In Section 2.3 we detail reasoning approaches which are used for semantic inference proving / matching. In Section 2.4 we review current matching approaches for mobile environments and categorise these based on their support for semantics and on-board mobile matching. Finally, in Section 2.5 we summarise the chapter.

[4]http://www.w3.org/TR/owl-features/ (accessed May 2009)

2.2 Services Matching Mechanisms

In this section we will provide an overview of semantic approaches for matching. In Section 2.2.1 we will provide an overview of the Semantic Web languages. Then in Section 2.2.2 we will provide an overview of the approaches used to describe Web Services semantically using these Semantic Web languages.

2.2.1 Semantic Web

As discussed in the previous section, a mechanism is required to select a service from the available pool of potentially many services, by matching these with the requirements of the user. Typical mechanisms for achieving this involves using string based keywords or key and value pairs, to compare services against requests (Raman et al., 1999). For example, a user specifies a request as a set of keywords or phrase which is then matched to keywords about a service using string comparison. This can produce inaccurate results due to the existence of synonyms and homonyms (Bernstein and Klein, 2002). It is challenging for a matching mechanism to completely and always understand the user's real intent with keyword input. A single word often has several meanings depending on the context in which the word is used (Peng et al., 2008).

Another typical mechanism matches services by interface, also known as table based matching (Bernstein and Klein, 2002). Moreover, a user request is defined in terms of required inputs and outputs of a service and the data types of each. For instance, with respect to Web Services, the Web Service Description Language (WSDL)[5] is used to describe the inputs and outputs of a Web Service using XML data types. These inputs and outputs can be used as part of the matching process to match service capabilities / functionality with a user request. These approaches are generally useful when the user has previous knowledge of the service and wishes to discover which device the service is located on, in order to connect to it. However, in extremely dynamic

[5]http://www.w3.org/TR/wsdl (accessed May 2009)

environments where the user is unlikely to have used the service previously, interface based matching does little to improve accuracy over keyword based approaches because it does not capture what information is expected by inputs or provided by outputs beyond data type. For instance, an input which has a string data type does not reveal much about service functionality. In addition, services with different goals may share similar service interfaces (Bernstein and Klein, 2002).

Semantic Web (SW) languages provide a more declarative mechanism for describing services in terms of their functionality and behaviour using semantics. Improved accuracy for service matching with user requests can, therefore, be achieved by adopting a semantic language to describe services and user requests (Abramowicz et al., 2008; Bernstein and Klein, 2002). The Semantic Web[6] (Sheth et al., 2005) is designed to support the classification of data for machine processing (Palmer, 2001) and has been supported by many industry partners including Sun Microsystems, Hewlett Packard, IBM and Nokia[7]. The first semantic language was the XML based Resource Description Framework (RDF)[8] language which classifies data into a graph structure of nodes and relations. The Web Ontology Language (OWL)[9] was built on RDF to provide increased expressiveness and definition of constraints. An OWL document is called an ontology (Singh and Huhns, 2005). Ontologies can be used by software applications to infer new information from existing data (Palmer, 2001) which may have extended relationships across many ontologies. The OWL language provides three sublanguages, each with increasing expressiveness: OWL-Lite, OWL-DL and OWL-Full, respectively. OWL-Lite enables the classification of a hierarchy and supports simple constraints (e.g. cardinality constraints of 0 and 1 only). OWL-Lite is the least expressive of the OWL languages and it is easier to provide tool / application support for OWL-Lite

[6]http://www.w3.org/2001/sw (accessed May 2009)
[7]http://www.w3.org/2004/01/sws-testimonial (accessed May 2009)
[8]http://www.w3.org/RDF/ (accessed May 2009)
[9]http://www.w3.org/2004/OWL (accessed May 2009)

than for the other OWL sublanguages and computations using OWL-Lite are more efficient. OWL-DL corresponds to Description Logic (DL) (Baader et al., 2005) and "offers the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time)" (W3C, 2004). Moreover, OWL-DL is more expressive than OWL-Lite and tools / applications can still complete computations for every feature of OWL-DL. Finally, OWL-Full offers the maximum level of expressiveness but has no computational guarantees. In other words, it is possible to create definitions using OWL-Full which software applications cannot compute a result about (W3C, 2004). Each sublanguage is an extension of its simpler predecessor (e.g. OWL-DL is an extension of OWL-Lite).

The question that arises is that given ontological descriptions of services and requests, how can this information be leveraged to perform matching. The simplest approach is to use an ontology as a mechanism for classification of terms to create a taxonomy. Examples of such taxonomies include WordNet[10], OpenCyc[11], DOLCE[12] and SUMO[13]. Ontology data can be queried using the Semantic Protocol and RDF Query Language (SPARQL)[14] as proposed in Bernstein and Klein (2002). Another approach is to compare terms in an ontology using distance metrics such as edge counting (Jiang and Conrath, 1997; Resnik, 1999) or probabilistic models (Lin, 1998) as proposed in Doulkeridis and Vazirgiannis (2008); and Hliaoutakis et al. (2006). These approaches are extremely efficient to compute a result, however, they do not take complex constraints into consideration which can be used to infer new information (Bernstein and Klein, 2002).

[10]http://wordnet.princeton.edu/perl/webwn (accessed May 2009)
[11]http://www.opencyc.org (accessed May 2009)
[12]http://www.loa-cnr.it/DOLCE.html (accessed May 2009)
[13]http://www.ontologyportal.org (accessed May 2009)
[14]http://www.w3.org/TR/rdf-sparql-query (accessed May 2009)

Alternatively, software applications can be used to make inferences using semantic knowledge. These are called semantic inference provers or semantic reasoners (Baader et al., 2003, p. 20). An inference is the act of deriving new information from the facts which are explicitly defined (Palmer, 2001). This involves checking whether a particular graph of data meets a certain constraint definition. Most current OWL reasoners support OWL-DL, including the open source and commercial semantic reasoners such as Pellet[15], KAON2[16], FaCT++[17] or RacerPro[18]. Semantic reasoners are required in order to make effective use of the full expressiveness supported by OWL-DL (Baader et al., 2005). Therefore, in order to provide mobile users with accurate service matching our thesis will focus on describing services using OWL-DL and develop an efficient semantic reasoner to perform inference based matching on a mobile device.

In the next section we will provide an overview of frameworks which are designed specifically for describing Web Services using semantics.

2.2.2 Semantic Web Services

In terms of matching user requests against service descriptions, there are a number of languages which are designed specifically for describing services using semantics. One approach is to use OWL Services (OWL-S) (McIlraith et al., 2001) upper ontologies which have been designed for describing services using OWL. OWL-S can be used to describe a service at three different levels:

- **Service Profile:** which uses semantic concepts to describe a service in terms of its type, its functional properties including inputs, outputs, preconditions and results (IOPRs) and non-functional properties such as quality of service;

[15]http://clarkparsia.com/pellet (accessed May 2009)
[16]http://kaon2.semanticweb.org (accessed May 2009)
[17]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[18]http://www.racer-systems.com (accessed May 2009)

- **Service Model:** which is used to compose several services into larger composite services and processes;

- **Service Grounding:** which is used to bind the semantic service profile to an actual concrete, invokable Web Service.

An alternative approach for describing services is the Web Service Modelling Ontology (WSMO)[19], which provides the Web Service Modelling Framework (WSMF) (Fensel and Bussler, 2002) comprising four main elements:

- **Ontologies:** which is used to describe the terminology used by the Goals, Web Service Descriptions and Mediators;

- **Goals:** which is used to define problems which the Web Service solves or define user objectives that a potential service should solve;

- **Web Service Descriptions:** which is used to define the semantic description of the Web Service

- **Mediators:** which is used to define the logic expressions and rules to handle interoperability such as connecting different concepts in the Ontologies, different Goals, different Web Services Descriptions and Goals to Web Service Descriptions.

Another alternative for describing services is to use the WSDL-S[20] language. It is designed as a light-weight approach, adding semantic annotations to properties such as inputs and outputs of a Web Service, described in its WSDL document. Thus WSDL-S is focused on describing a service as an operation.

Services can also be described using the Semantic Web Services Framework (SWSF) which is made up of the Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO). This framework is concerned

[19]http://www.wsmo.org (accessed May 2009)
[20]http://www.w3.org/Submission/WSDL-S (accessed May 2009)

with the specification of a Web Service using First Order Logic (FOL) and rules based on the logic programming paradigm. Like OWL-S, it provides a Service Profile, Process Model and Grounding.

The Semantic Annotations for WSDL (SAWSDL)[21] describes services by extending WSDL to support semantic annotations, using any semantic language. Unlike WSDL-S, it does not address service annotations to service implementations including bindings, services and end-points.

The various Semantic Web Services (SWS) frameworks described in this section are designed for describing an actual invokable service. These approaches generally describe a service as a set of constraints which can be used to specify service types, inputs and outputs. Some of these frameworks also make use of the First Order Logic (FOL) based languages such as Semantic Web Rule Language (SWRL)[22] to specify the preconditions and effects of services. Our research is focused on OWL constraint matching, and rule languages are not the focus of our research. In addition, while we focus on constraint matching in OWL to match user requests against service descriptions, we do not restrict our approach to only matching of SWS service descriptions such as OWL-S profiles, which are designed to represent only concrete invokable services. Though this kind of matching is supported by our approach, we also wish to support semantic matching of service descriptions beyond those that are invocable (e.g. goods in a supermarket). To illustrate further, a mobile user may wish to locate a shopping centre which has a children's playground for his/her child to play on while he/she shops. This is not a Web Service but a "real world" centre. It is conceivable that such descriptions with the most up-to-date content can be easily delivered to mobile devices via network communications such as short range bluetooth or WiF, as previously described in Chapter 1. Therefore, our approach aims to support a broad range of semantic matching.

[21] http://www.w3.org/2002/ws/sawsdl (accessed May 2009)
[22] http://www.w3.org/Submission/SWRL/ (accessed May 2009)

In this section, we have identified the need to support semantic matching using OWL-DL (Baader et al., 2005) descriptions, in order to achieve a high level of accuracy in matching. However, a decision procedure / inference proof algorithm is required to use these languages in order to prove or disprove inferences about knowledge, described using OWL-DL. These algorithms are implemented in applications known as reasoners. We now provide an overview of reasoning approaches using semantics in the following section.

2.3 Reasoning Approaches

In this section we will provide an overview of different reasoning algorithms and associated logics used to prove or disprove inferences.

Propositional logic (O'Donnel et al., 2006, p. 109) allows the specification of truth toward certain objects or combinations of objects. For instance, using propositional logic, it is possible to write a statement defining that a vehicle is a car or truck. First Order Logic (FOL) (Galton, 1990; Kelly, 1997) extends propositional logic, to add relationships between objects, which are also called nodes in FOL, and adds the ability to define quantified constraints over these relationships. In FOL a relationship, which is also known as a predicate (Kelly, 1997), can have any number of arguments. For instance, using FOL it is possible to write a sentence defining that a car is something that has doors and four wheels. FOL is an undecidable logic (Church, 1936), meaning that a solution cannot always be found.

Due to the fact that FOL is undecidable, some decidable subsets have been identified. One such subset is known as Description Logic (DL) (Baader et al., 2003) which supports sound and complete reasoning. DL restricts predicates (known as roles in DL (Baader et al., 2003)) to atomic unary predicates only (i.e. they can only have one argument) (Kelly, 1997). DL can have various levels of expressiveness, using the informal naming convention as follows. The most basic possible level is called Attributive Language $\mathcal{AL}$. This basic

language supports atomic concepts, negation, intersection, universal quantification and limited existential quantification. Other constructors can be added for extra expressiveness including $\mathcal{U}$ union, $\mathcal{E}$ full existential quantification, $\mathcal{N}$ number restrictions and $\mathcal{C}$ compliment (negation). The level of expressiveness supported is written using the notation: $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}]$. As described previously in Section 2.2.1, OWL-DL is based on Description Logic (DL). The Semantic Web languages are inspired by DL. OWL-DL version 1.1[23] is equivalent to DL $\mathcal{SHOIN}^{(\mathcal{D})}$, which is an alternative naming convention. In $\mathcal{SHOIN}^{(\mathcal{D})}$, $\mathcal{S}$ is an abbreviation for $\mathcal{ALC}$, with role transitivity, $\mathcal{H}$ role hierarchy (subsumption), $\mathcal{O}$ nominals (value restrictions), $\mathcal{I}$ inverse roles, $\mathcal{N}$ (unqualified) number restrictions and $(\mathcal{D})$ datatype properties containing data values (literals) and data types. OWL 2.0[24] which was made a WC3[25] recommendation in November 2009, supports DL $\mathcal{SHOIQ}$ which adds $\mathcal{Q}$ qualified number restrictions.

Since Description Logic is decidable, there are a number of reasoners which support DL inference. Early reasoners, which were based on structural DL subsumption algorithms, were efficient (Borgida and Patel-Schneider, 1994; Heinsohn et al., 1994) but incomplete (Baader et al., 2003, p. 80) and could not handle arbitrary knowledge such as ontologies (Doyle and Patil, 1991). This was because they compared the syntactic structure of two concepts rather than providing a resolution based decision proof to compare the semantic interpretation of these concepts. Examples of structural provers include Classic (Patel-Schneider et al., 1991), Loom (MacGregor, 1991) and Grail (Rector et al., 1997). Alternatively, complete and sound inferences can be achieved using the Tableaux (Calvanese et al., 2001; Horrocks and Sattler, 2007) decision resolution procedure, which takes account of semantic interpretation. It

[23]http://www.w3.org/TR/2004/REC-owl-features-20040210/ (accessed May 2009
[24]http://www.w3.org/TR/owl2-overview/ (accessed February 2010)
[25]http://www.w3.org/ (accessed February 2010)

attempts to prove an inference by refuting it and demonstrating that a consistent interpretation of the logic can no longer be generated. The Tableaux algorithm "has dominated recent DL research" (Baader et al., 2003, p. 322) because it has desirable properties of soundness (Hollunder et al., 1990) and completeness (Baader et al., 2003, p. 87), it ensures termination (Baader and Hanschke, 1991; Buchheit et al., 1993; Sattler, 1996) with an arbitrary set of ontologies, and is an efficient resolution procedure (Hollunder et al., 1990). It is efficient because it minimises the amount of space used by utilising an expansion tree and employs various optimisation strategies (Tsarkov et al., 2007; Horrocks and Patel-Schneider, 1999). Tableaux is implemented widely in many semantic reasoners such as Pellet[26], FaCT++[27], RacerPro[28].

Another known FOL subset which makes DL decidable are Horn clauses (Horn, 1951). FOL Horn clauses are rule based approaches also known as logic programs, which differ from DL constraint resolution which is used with OWL-DL (Grosof et al., 2003). DL constraint resolution is concerned with determining whether there is an inferred relationship between class concept definitions and whether a node is a type of a particular class. Conversely, rule based approaches such as FOL Horn, allow the specification of rules which suggest that some goal or conclusion is true if the rule conditions are met. The Semantic Web Rule Language (SRWL)[29] is a rule language for the Semantic Web. These FOL reasoners implement one of two kinds decision resolution procedures known as either backward chaining or forward chaining. In backward chaining (Smith et al., 1986) the reasoner is asked if a particular goal is true, and constructs a backtracking proof tree to check each condition of the goal. Prolog[30] and Lisp (Steele, 1990) are examples of backward chaining reasoners. In forward chaining, rules are specified and if the conditions of any rule are

[26]http://clarkparsia.com/pellet (accessed May 2009)
[27]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[28]http://www.racer-systems.com (accessed May 2009)
[29]http://www.w3.org/Submission/SWRL/ (accessed May 2009)
[30]http://www.swi-prolog.org (accessed May 2009)

met, a fact is either added or removed from the knowledge base. The RETE (Forgy, 1982) algorithm is generally used as the forward chaining inference proof. Jess[31] and Clips[32] are examples of forward chaining reasoners which use RETE. More recently, with the development of the Semantic Web, some Semantic Web reasoners support Horn FOL reasoning. The Jena[33] RDF inference prover supports both forward and backward chaining using its own rule language. In addition, both Pellet[34] and KAON2[35] support forward chaining using the SWRL language using RETE, in addition to OWL constraint inference resolution.

Datalog (Ceri et al., 1989) has been used as a DL decision resolution procedure (Hustadt et al., 2007) for OWL-DL. Datalog is a forward chaining query and rule language for deductive databases that syntactically is a subset of Prolog. It is a query language for relationship databases based on first-order logic (Ajtai, 1989). Datalog supports Horn clause logic and is a simplified version of general Logic Programming (LP) (Lloyd, 1987). Datalog employs strategies for efficient reasoning when querying large scale database data since Prolog[36] is a Horn clause reasoner, which we will discuss later in this section. KAON2[37] (Hustadt et al., 2004) is an OWL-DL reasoner which reduces reasoning tasks to a disjunctive Datalog (Eiter et al., 1997) program. This allows for the application of disjunctive database techniques and optimisations to DL reasoning.

As described in Section 2.2.2 some Semantic Web Service approaches, utilise rules for expressing the preconditions and effects of a particular Web Service. Rules are supported as an addition to the support for DL constraint based matching which determines whether a particular node in an ontology

[31] http://www.jessrules.com (accessed May 2009)
[32] http://clipsrules.sourceforge.net/ (accessed May 2009)
[33] http://jena.sourceforge.net/ (accessed May 2009)
[34] http://clarkparsia.com/pellet/ (accessed May 2009)
[35] http://kaon2.semanticweb.org/ (accessed May 2009)
[36] http://www.swi-prolog.org (accessed May 2009)
[37] http://kaon2.semanticweb.org (accessed May 2009)

meets the constraints specified in a class concept definition. As stated in Section 2.2.1, in this thesis we are addressing the need to support DL constraint resolution on-board small mobile devices, to support a broad range of service matching, rather than rule resolution. In terms of DL resolution, the Tableaux algorithm is the most commonly used decision procedure (Baader et al., 2003, p. 322) used by open source and commercial reasoners including Pellet, RacerPro and FaCT++. However, as discussed Section 1.2, despite its efficiency, the Tableaux algorithm is considerably resource intensive (Zacharias et al., 2007; Fensel et al., 2008). Therefore, the Tableaux algorithm in its current operational state does not scale to small resource constrained mobile devices as we determined through our preliminary testing / evaluation (see Figure 1.2 in Section 1.2). While accuracy and completeness of results is an important consideration to reasoners, performance efficiency is equally important in mobile environments.

Having discussed the different logics and reasoning approaches, in the next section, we will provide an overview of the current techniques used for service matching.

2.4 Semantic Service Matching in Mobile Environments

In this section we describe current techniques which support service matching.

We will categorise these techniques in terms of whether they support semantic matching for greater accuracy / reliability over simple matching such as keyword based approaches and whether they support efficient / fast matching deployable on a small resource constrained mobile device or require a static / high-end server based node to perform matching. These approaches can be categorised using the taxonomy presented in Figure 2.1.

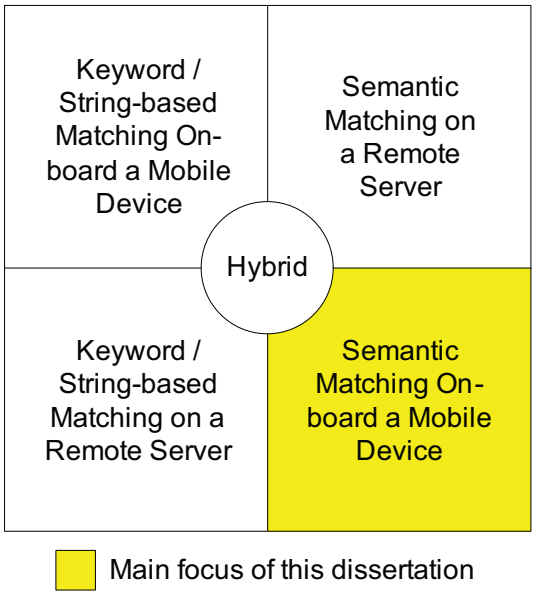


Figure 2.1: Categorisation of current matching approaches

As shown on the left vs the right side of this figure, matching approaches generally utilise either keyword / interface based matching mechanisms or semantic inference based matching which implies the use of a semantic language such as those presented in Section 2.2.1 and a reasoner to make inferences on the semantics. As we discussed Section 2.2.1, semantic inference matching is more accurate than simple keyword or interface matching. In addition, we categorise each of these approaches into those where the matching is completed using:

1. a remote / external server, which performs matching on behalf of the resource constrained device. Typically these are centralised statically deployed servers. This may also include desktop PCs or laptops;

2. an on-board a mobile device, which are typically resource constrained;

3. a hybrid approach, which typically requires that processing is pre-emptively completed on a high-performance machine before matching can then take place on a resource-constrained mobile node.

The focus of this dissertation is to support semantic matching on-board a mobile device, which is highlighted in Figure 2.1. We will now describe current service matching techniques according to the categories presented in the figure.

2.4.1 Keyword / Interface Matching Using Remote Servers

Early architectures for service matching which use keyword / interface matching include Jini (Arnold et al., 1999; Helal, 2002); Service Location Protocol (SLP) (Guttman, 1999; Helal, 2002), Salutation (Helal, 2002), UDDI[38] and LDAP (Howes and Smith, 1995). These architectures support string-based key and attribute matching only, and operate using a static centralised directory node to perform matching.

2.4.2 Keyword / Interface Matching On-board a Mobile Device

There are various approaches (Su and Guo, 2008) which support simple matching which can be deployed on-board a mobile device in a distributed environment. Service Location Protocol (SLP) (Guttman, 1999; Helal, 2002), and Salutation (Helal, 2002) can also operate in a peer-to-peer (P2P) environment without a static node. UPnP[39] (Helal, 2002) operates in a peer-to-peer environment and supports keyword matching.

Ratsimor et al. (2002); Koodli and Perkins (2002); and Ververidis and Polyzos (2005) simply compare services based on a Universal Unique Identifier (UUID) assigned to them. Shim et al. (2009); and Liang et al. (2007) convert service descriptions into a 128-bit integer using a hash function and compare these integers for service matching. Islam and Shaikh (2008) match using service name and type. Konark (Lee et al., 2003) utilises keywords and a Web

[38] http://uddi.xml.org/ (accessed May 2009)
[39] http://www.upnp.org/ (accessed May 2009)

Service Description Language (WSDL) defined service interface for matching. SSD (Sailhan and Issarny, 2005) also performs string and WSDL interface matching along with Quality of Service (QoS) metrics. Blefari-Melazzi et al. (2007) compares WSDL interfaces and the required device capabilities of the service. LSD (Li and Lamont, 2005) compares services based on location using a native Service Location Extension (SLE) message.

Srirama et al. (2008) performs matching based on keywords and categorisations similar to UDDI, including the North American Industry Classification System (NAICS) and United Nations Standard Products and Services Code (UNSPSC). PDP (Campo et al., 2006) associates each service with a type, defined in a native classification hierarchy and matches based on these types.

The benefit of these approaches is that no static or remote infrastructure is required. The matching process occurs on-board the mobile devices themselves and uses the information gathered from other devices within network range. The disadvantage is that these approaches do not support semantic reasoning which is more accurate than keyword / interface based matching.

In the next section we will discuss semantically driven approaches which utilise reasoners to perform semantic inferences while matching.

2.4.3 Semantic Inference and Matching Using a Remote Server

There are a number of early approaches that support semantics but typically either require a static or remote, high-performance server to perform the inference matching and are, therefore, not suited for on-board mobile matching.

CoBrA (Chen et al., 2004) was one of the early middleware architectures which utilises semantics and context to reason about users and situations in smart ubiquitous spaces on a static node. The Task Computing Project (TCP) (Masuoka et al., 2003) utilises OWL-S for modelling services for smart meeting room scenarios. Integrated Global Pervasive Computing Framework (IGPCF)

(Singh et al., 2005) offers web service discovery using semantics on the web but assumes that pervasive users are permanently connected to the Internet.

Luo et al. (2005, 2006) adds OWL-S descriptions to the UDDI service registry which performs inferences when a service description is published to it. DReggie Chakraborty et al. (2001) extends Jini to support semantic matching using Prolog[40] for reasoning. LARKS (Sycara et al., 2002) is designed to match service descriptions with requests, using its own semantic description language. This approach uses matching mechanisms ranging from keyword matching through to distance based ontology comparisons and logic rule constraint evaluations. The matching mechanism used will depend on the level of accuracy required.

The CMU Matchmaker (Srinivasan et al., 2005) provides inference based reasoning to compare OWL-S service profiles with requests, which can be stored in a back-end UDDI registry. Matching is based on service type and the inputs and outputs of the service, using RacerPro[41] to perform inferences. Bener et al. (2009) proposes a matchmaking algorithm which also takes preconditions and effects into consideration using SWRL. Stuckenschmidt and Kolb (2008) defines a reasoning approach which supports partial matching of services against requests where there is insufficient time to complete the full matching process. However, this is achieved by reducing the number of conditions in the request. Matching is still performed on a standard reasoner such as Pellet.

Semantic Web Engineering - Environment and Tools (SWE-ET) (Brambilla et al., 2006) combines the CEFREIEL Glue[42] discovery engine with the WebRatio[43] framework to support WSMO Semantic Web Service discovery. The SWE-ET supports semantic reasoning using F-logic (Kifer et al., 1995)

[40]http://www.swi-prolog.org/ (accessed May 2009)
[41]http://www.racer-systems.com (accessed May 2009)
[42]http://glue.cefriel.it (accessed May 2009)
[43]http://www.webratio.com (accessed May 2009)

supported by Flora-28[44] with the XSB inference engine[45] based on Prolog, which offers flexibility to handle a wide range of scenarios. The Internet Reasoning Service (IRS)-III [46] (Domingue et al., 2008) is a Semantic Web Service broker and reasoning environment which is again based on WSMO but has the added functionality of importing OWL ontologies. The brokering server runs a HTTP Lisp reasoner (Riva and Ramoni, 1996) and utilises both forward and backward chaining rules. DIANE[47] (Kuster and König-Ries, 2008) is an environment for automated service discovery and matching which uses its own service profile language to describe a service as a set of effects. It supports a subset of logic without any rules or quantifiers and provides "fuzzy" matching of conditions in the user request against the service description, to provide ranked service results.

There are also architectures which have been designed specifically for mobile clients, but require a remote server to perform matching. Broens et al. (2004); and Doulkeridis and Vazirgiannis (2008) utilise semi-OWL and RDF documents to express service descriptions and perform matching on a centralised server repository such as UDDI[48] or ebXML[49]. Baousis et al. (2008); de Andrade et al. (2007); and Chen et al. (2006), support matching of mobile services, but rely on the CMU matchmaker (Srinivasan et al., 2005) described earlier in this section, which uses a UDDI server back-end. Jeon et al. (2008) matches semantically described personal preferences using OWL and SWRL roles on an external server. Suraci et al. (2007); and Srirama et al. (2007) support OWL-S service matching on a high-end node. Bianchini et al. (2006) is an approach which provides UDDI based matchmaking of semantically described services in a mobile environment based on location and device capabilities.

[44]http://flora.sourceforge.net (accessed May 2009)
[45]http://xsb.sourceforge.net (accessed May 2009)
[46]http://technologies.kmi.open.ac.uk/irs/ (accessed May 2009)
[47]http://hnsp.inf-bb.uni-jena.de/DIANE (accessed May 2009)
[48]uddi.xml.org (accessed May 2009)
[49]http://www.ebxml.org (accessed May 2009)

Veijalainen et al. (2006) performs mobile semantic service matching. However, matching was performed on laptops which are not resource constrained (as opposed to PDAs / mobile phones). Wang and Hu (2008) is a P2P semantic OWL-S matching architecture which attempts to reduce the number of inference checks required, by performing an initial keyword search to limit the number of services which are semantically matched. Niazi and Mahmoud (2009); Wolowski et al. (2007); Zoric et al. (2007a); El-Sayed and Black (2006); Almeida et al. (2006); and Sycara et al. (2002) matches services described in OWL using the Jena[50] forward chaining inference engine. Peng et al. (2008) delegates OWL service matching to a resource capable machine and uses RacerPro[51] to perform all reasoning. AIDAS (Toninelli et al., 2008) performs matching of mobile user preferences and device capabilities for services using the Pellet reasoner. Gaia (Ranganathan and Campbell, 2003) is a semantically driven context mobile middleware which performs matching utilising the FaCT++[52] reasoner. Patel and Chaudhary (2009); De and Moessner (2008); and Wei et al. (2008) support semantic queries and matching by making use of Jess[53] which is a forward chaining First Order Logic (FOL) reasoner.

Agostini et al. (2007); Mokhtar et al. (2008); and Bouillet et al. (2008) perform all inferences offline on a high-performance server, before matching is later completed on-board a resource constrained device. These approaches can be considered hybrids, but since they require a high-performance server, we have listed them in this section.

All of the semantically driven approaches described in this section have been shown to operate on a high-performance machine or require such a machine to perform pre-processing before the matching occurs. Most of the approaches utilised a reasoner such as the Jena inference prover, Jess, RacerPro, FaCT++, Pellet, Prolog or Lisp. The use of these reasoners is typically a

[50]http://jena.sourceforge.net/ (accessed May 2009)
[51]http://www.racer-systems.com/ (accessed May 2009)
[52]http://owl.man.ac.uk/factplusplus/ (accessed May 2009)
[53]http://www.jessrules.com/ (accessed May 2009)

resource intensive operation. More recently, the HermiT[54] (Motik et al., 2009) reasoner has been developed to provide more optimised Tableaux semantic DL reasoning using OWL. However, like other similar reasoners such as RacerPro, FaCT++ and Pellet, HermiT has been developed for the desktop / server environment. None of the approaches described in this section performed semantic inference matching on-board a resource constrained mobile device.

2.4.4 Semantic Inference and Matching On-board a Mobile Device

In this section we review matching approaches which both support semantics and operate using a mobile device. Chakraborty et al. (2006); Nedos et al. (2006) are approaches which match services based on semantic service types defined in a hierarchy. However these approaches use explicit subclass relations only (such as OWL-Lite) and do not support semantic reasoning and inference proof. As we described in Section 2.3, semantic reasoners are software applications which implement a decision procedure to determine whether or not a semantically described service matches a user request. In the context of mobile users, the semantic matching which aims to enhance accuracy must be supplemented by fast response times as well. As we described in Section 1.2, current open source and commercial reasoners such as Pellet[55], KAON2[56], FaCT++[57] or RacerPro[58] are considerably resource intensive (Zacharias et al., 2007; Zoric et al., 2007b) and thus cannot function on small resource constrained devices. This was illustrated previously in Figure 1.2 in Section 1.2, where we attempted to perform matching of a user request to locate a printer against a semantic service description, using the standard Pellet reasoner on

[54]http://hermit-reasoner.com/ (accessed May 2009)
[55]http://clarkparsia.com/pellet (accessed May 2009)
[56]http://kaon2.semanticweb.org (accessed May 2009)
[57]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[58]http://www.racer-systems.com (accessed May 2009)

a HP iPAQ PDA. The matching process failed to complete because there was insufficient memory available.

These issues of resource constraints have motivated an emerging body of research in developing reasoners for mobile devices. Therefore, we review mobile reasoning approaches which are used for matching a semantically described service with a user request.

Kleemann and Sinner (2006) is an approach to service matching which uses KRHyper (Kleemann, 2006), which is a novel Tableaux reasoner for First Order Logic (FOL) for deployment on resource constrained devices. In order to use DL with KRHyper it must be transformed into a set of disjunctive first order logic clauses. It implements the standard Tableaux optimisation strategies of backjumping, semantic branching, Boolean constraint propagation, lazy unfolding and absorption (Tsarkov et al., 2007), used by today's commercial and open source reasoners. Performance comparisons show that KRHyper provides performance which is compariable to RacerPro[59] but tends to provide the same or better response time for test cases which contained 10 or fewer subsumption checks, and did not perform as well for larger tests (Kleemann, 2006). All tests had short branch sizes with less than 25 Tableaux proof steps, because the test ontology was not deeply nested. KRHyper still exhausts all memory when the reasoning task becomes too large for a small device to handle and fails to provide any result.

Ruta et al. (2008a,b) supports distance based, ranked, matching of requests to services using a DL mobile reasoner implemented in Java 2 Micro Edition (J2ME)[60] which supports short range (bluetooth) ad-hoc networks. It achieves acceptable performance by restricting the OWL-DL language to a subset. Additionally, the ontology structure is constrained so that it can be reduced to set comparisons in order to downscale the computational demand of inference algorithms to meet the capabilities of handheld computing devices.

[59]http://www.racer-systems.com (accessed May 2009)
[60]http://java.sun.com/javame/index.jsp (accessed May 2010)

Gu et al. (2007) is a framework which provides an RDF/OWL parser, reasoner and sRDQL query engine for information matching such as a shopping assistance application which uses a user's context to provide suggestions to the user about products which may suit their needs based on previous usage patterns. This framework runs on the user's mobile device using J2ME. This framework provides acceptable performance by supporting a subset of the OWL language known as OWL-Lite[61] as mentioned in Section 2.2.1. The reasoner in the framework uses a forward chaining (see Section 2.3) approach which supports rule based reasoning, such as SWRL[62]. As mentioned in Section 2.3, in this thesis we are addressing the need to support DL constraint resolution on-board small mobile devices, to support a broad range of service matching, rather than rule resolution.

As we outlined in Section 2.2 our main focus is on matching semantically described service descriptions against user requests, using a mobile inference prover / reasoner to provide a greater level of accuracy, compared to simple matching techniques such as keyword or interface matching. There is a growing emphasis on this need which has given rise to several mobile reasoning approaches. The main drawback of these approaches was that in the case of Ruta et al. (2008a); and Gu et al. (2007), acceptable performance on a resource constrained mobile device was only achieved by restricting the OWL language to a subset, to reduce the resource demands on the device, such as proposed in Hitzler and Vrandecic (2005). KRHyper support the full range of OWL-DL constructs and provides accurate results. It does not introduce new optimisation strategies beyond those implemented by current DL reasoners and is, therefore, prone to exhaustion of available memory when reasoning with larger ontologies. Therefore, the above approaches for mobile semantic reasoning either:

[61] http://www.w3.org/TR/owl-features/ (accessed May 2009)
[62] http://www.w3.org/Submission/SWRL/ (accessed May 2009)

1. use OWL-Lite or another subset of the OWL-DL language, thereby supporting reduced expressiveness compared to OWL-DL;

2. use the standard Tableaux in a mobile setting, which still exhausts all memory when reasoning with large ontologies.

However, reasoner support for OWL-DL achieves a greater level of matching accuracy when compared to using subsets of OWL-DL, because these subsets are less expressive. As such, there is a need for an OWL-DL reasoning approach which is efficient / light-weight and therefore scalable for a mobile device. While accuracy is important, efficiency must also be considered as an important goal. Therefore, in the remainder of this thesis we propose and develop optimisation strategies to provide more efficient reasoning for OWL-DL on a mobile device without significantly reducing accuracy. In addition, we propose and develop an adaptive inference strategy which takes account of constraints such as user preferences, time and resources during the matching process.

2.5 Summary

In this chapter we have reviewed various approaches for matching service descriptions with user requests. Keyword and interface based matching are less accurate when compared with semantic based approaches because it is very difficult to capture the user's intent using keywords. Alternatively, semantic descriptions capture the functionality and behaviour of services using semantics. OWL-DL[63] is a semantic language which is based on Description Logic (DL) and it provides the maximum expressiveness without losing computational completeness and decidability. Semantic reasoners allow new information to be inferred from the semantic descriptions so that service descriptions can be matched with user requests, even if these are described using a different

[63]http://www.w3.org/TR/owl-guide/ (accessed May 2009)

syntax. However, semantic reasoning is considerably resource intensive and a review of current techniques, which match user requests with services, has revealed that current semantic matching techniques generally delegate matching to a static / high-end server. There is an emerging body of research focusing on the development of mobile reasoners. However, current mobile reasoners either support only a subset of OWL-DL, such as OWL-Lite, thereby reducing expressiveness or exhaust the available memory when reasoning with large ontologies. Thus, there is no reasoner which enables efficient / light-weight reasoning on a mobile device for large ontologies while maintaining OWL-DL expressiveness. Therefore, in this thesis will propose and develop strategies to:

- enable efficient / light-weight reasoning on a mobile device without significantly reducing accuracy;
- support adaptive reasoning so that the reasoning process can be interrupted based on user constraints such as available time or resources and provide a match result based on the computations performed up to the point of interruption.

In order to understand our strategies, we must first provide an overview of the necessary background of Description Logic (DL) and the Tableaux inference proof, which our strategies are based on. We will present this overview in the next chapter.

Chapter 3

Background

3.1 Introduction

As previously discussed in Section 1.3 we identified the need for efficient and accurate mobile service selection, where matching occurs on-board a small resource constrained device. In order to achieve a high level of accuracy, services and the user request they are matched against must be described semantically (Bernstein and Klein, 2002) using a language such as Web Ontology Language (OWL)[1].

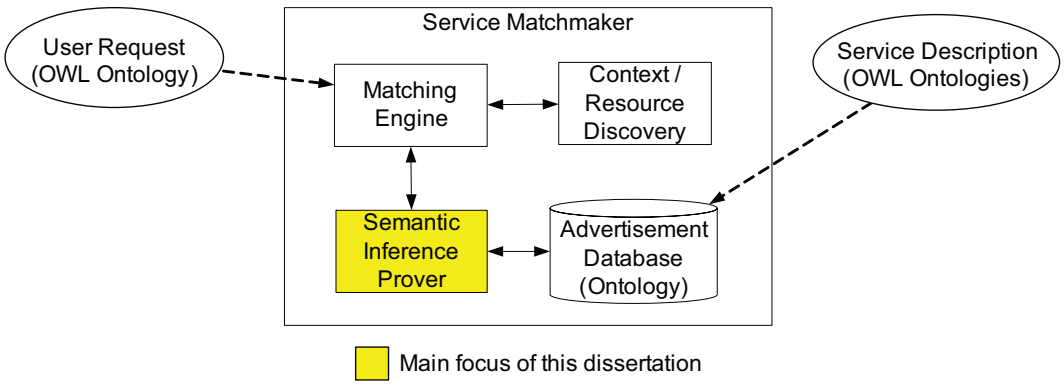


Figure 3.1: Generic Service Selection Model

Most semantic service matching architectures are a variation of the generic matchmaker shown in Figure 3.1 which is adapted from Paolucci et al. (2002).

[1]http://www.w3.org/TR/owl-features (accessed May 2009)

The Advertisement Database contains semantic service descriptions about advertised services. The Matching Engine coordinates the matching process. The Semantic Inference Prover performs the actual match check. Generally, architectures utilise an existing semantic inference prover such as Pellet[2], KAON2[3], FaCT++[4] or RacerPro[5]. Some architectures such as Doulkeridis and Vazirgiannis (2008) also utilise context, such as user preferences and device capabilities, as attributes during the matching process. These contextual attributes are obtained by the Context / Resource Discovery module. The focus of our contributions, outlined in this dissertation, is on the Semantic Inference Prover, which is highlighted in yellow in the figure.

Figure 3.2 illustrates the interaction between the components of the service selection process.

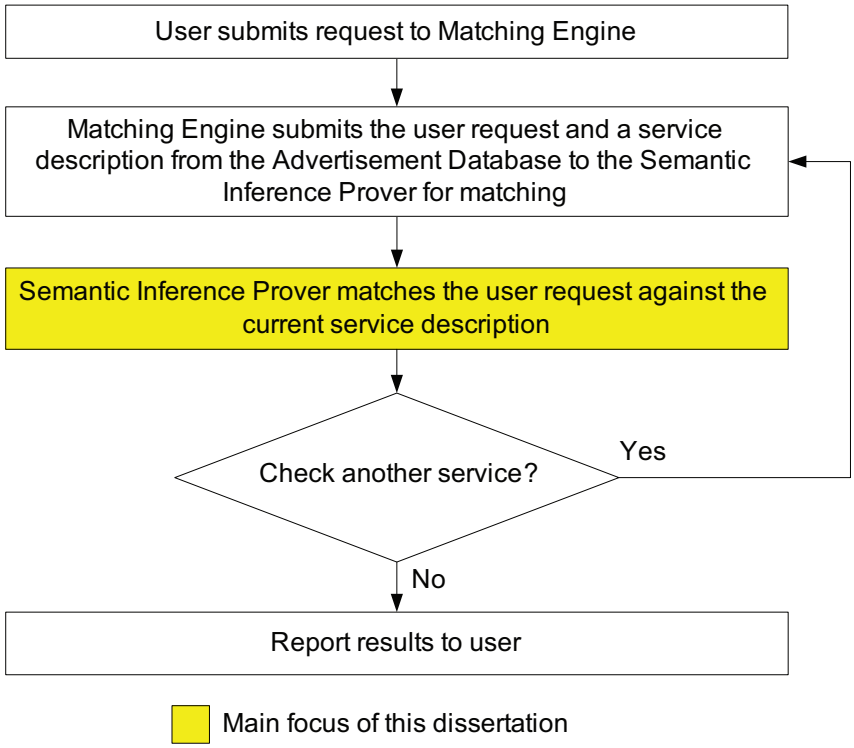


Figure 3.2: Generic Service Selection Component Interaction

[2]http://clarkparsia.com/pellet (accessed May 2009)
[3]http://kaon2.semanticweb.org (accessed May 2009)
[4]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[5]http://www.racer-systems.com (accessed May 2009)

The matching process begins when the user submits a semantic request to the matchmaker. The Matching Engine selects a service description from the database of advertised service descriptions and submits this description to the Semantic Inference Prover for matching against the request. The Semantic Inference Prover will check whether a semantic correspondence exists between the service description and the user request. After checking a particular service description, the matching engine will also determine whether or not there are more service description against the user request. Once all service descriptions have been checked, a ranked set of results for each service description is returned to the user.

In our approach, we propose that the Semantic Inference Prover will perform matching, taking into account the adequacy of the time / resources available to continue matching, before checking each user requirement / condition in the request. This operation is presented in Figure 3.3, which illustrates the sub-operations for the Semantic Inference Prover.

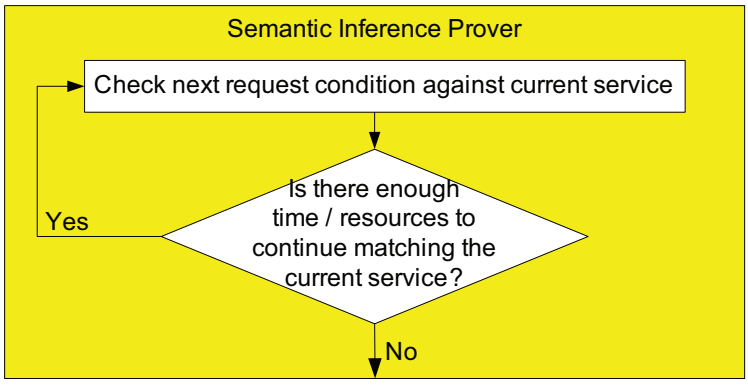


Figure 3.3: Proposed Adaptive Inference Prover Process

Most current Semantic Inference Provers are based on the Tableaux decision procedure decision proof which has “dominated DL research” (Baader et al., 2003, p. 322). Tableaux is used to prove or disprove an inference. This procedure constructs a search tree, in which inference matches are searched for. As discussed in Chapter 1, although Tableaux is considered to be a relatively efficient inference proof, current semantic reasoners are still far too resource

intensive and scale poorly to small resource constrained devices (Zacharias et al., 2007; Kleemann, 2006). These devices, typically lack sufficient memory or processing capabilities to perform semantic matching.

Therefore, in this thesis we propose and develop cost efficient optimisation strategies for the widely used Tableaux algorithm (Baader et al., 2003, p. 322) for semantic reasoning, to enable it to operate effectively on a mobile device. Our proposed optimisation strategies form mobile Tableaux (mTableaux) which enables mobile semantic reasoning on resource constrained devices. mTableaux is based on the premise that the constraint of completeness for reasoners can be relaxed, to obtain considerable performance improvements, without significantly reducing the validity of the inference process. In a mobile setting, the need for correct information is important but it is equally necessary for information to be provided with a quick response time. For instance studies such as Roto and Oulasvirta (2005), have shown that the users' attention shifts elsewhere after 10-15 seconds because mobile environments are typically very dynamic. Therefore, mTableaux enables mobile semantic reasoning by reducing completeness as a trade-off for efficiency. In addition, mobile users have certain interests, habits and preferences which influence what they will do in the future (Kurkovsky et al., 2005). This suggests that users may perform requests which are similar to previous requests they have performed. Most current semantic reasoners provide no persistent caching of previous inference checks (Zacharias et al., 2007). Therefore, in mTableaux the results from previous requests are stored in a cache so that they can be reused when the user performs similar requests in the future.

Finally, in order to provide a result, current reasoners must complete the entire matching process in full, otherwise no result is provided. In addition, they provide a result as either a successful or unsuccessful match, with no intermediate / incremental levels of matching. A "no match" result is provided in the case that any request condition in the inference check fails, even if that

particular requirement is not very important to the user. Therefore, to cater for resources and user requirements, another significant contribution of our research is to propose and develop an adaptive reasoning strategy, which enables "anytime" matching which has the innovative capacity to provide incremental matching of a request to a service, driven by constraints such as user preferences, time and availability of resources. Using our proposed approach, the match result given to the user is based on the extent of the processing completed. This ensures that a result is provided, even if it is less accurate due to there being insufficient time or resources available to complete the matching process in full.

In this thesis, to enable mobile semantic reasoning we propose and develop the following strategies:

Optimisation: This strategy aims to achieve improved computational performance as a trade-off with completeness of the inference checking. While accuracy is important, it is equally necessary in a mobile environment for information to be provided with a quick response time. We hypothesise that these strategies will provide significant performance gains while maintaining a high degree of accuracy / validity;

Caching: This strategy enables caching which stores the results of previous requests, so that these can be used in similar, future requests. Given the growth of secondary storage media, we incorporate caching mechanisms to semantic inference checks in order to improve response time;

Adaptive Inference: This strategy enables adaptive inference which supports incremental priority matching. Under this strategy an inference check may be interrupted prematurely based on user constrained such as insufficient resources or time available to continue matching. A weighted degree of match is provided to the user based on the computations / inference checks performed up to that point in time.

This chapter and the thesis focuses on providing readers with the necessary theoretical background for presenting the contributions of this research. This chapter is included for the purposes of improving clarity and comprehensibility of the main contributions of this research.

This chapter is organised as follows. Section 3.2 will provide an overview of semantic inference provers. In Section 3.3 we will provide an overview of relevant Description Logic (DL) notation. Section 3.4.2 details the Tableaux algorithm and its transformation / expansion rules which are used for inference proof. Section 3.5 details the process by which Tableaux transformation / expansion rules are executed using a branching search tree control mechanism. Our light-weight mTableaux optimisation and caching strategies apply to the Tableaux algorithm's transformation rules and will be outlined in Chapter 4. Our adaptive inference strategy applies to the Tableaux search tree and will be outlined in Chapter 5.

3.2 Semantic Inference Provers

The focus of this thesis is on semantic inference proof which is implemented by semantic reasoners. Semantic reasoners utilise semantic languages to prove or disprove a conjectured inference. Semantics can be represented using the Web Ontology Language (OWL)[6], which is serialised into Resource Description Framework (RDF)[7] documents represented in XML. The principle workflow of the semantic inference process is shown in Figure 3.4. The steps given in the figure are a generic representation and may differ depending on specific reasoner implementations. For instance some reasoners may combine one or more steps.

OWL ontologies which are stored as RDF/XML serialised files must be parsed into main memory using an RDF/XML Parser. Examples of publicly

[6]http://www.w3.org/TR/owl-features (accessed May 2009)
[7]http://www.w3.org/RDF (accessed May 2009)

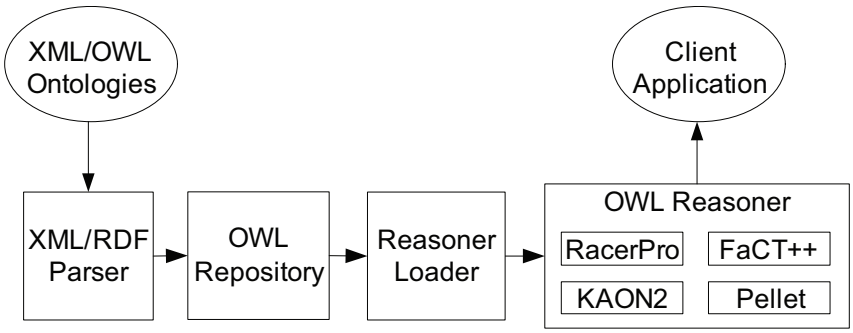


Figure 3.4: Generic Semantic Reasoner Process

available RDF Parsers include the ARP RDF Parser provided with Jena[8] and the RDF Parser provided with OWL-API[9]. The parsed RDF documents are stored in an OWL Repository, which allows the OWL constructs to be queried from main memory. Jena and OWL-API both include OWL repositories in their implementation. However, an OWL repository provides querying of explicit data only. Therefore, in order to draw new inferences from the data it needs to be loaded into reasoner. The Reasoner Loader iterates through all of the data contained in the OWL Repository and loads these into the OWL Reasoner's knowledge base. There are several different open source and commercial reasoners which may be used to perform reasoning on ontologies including Pellet[10], KAON2[11], FaCT++[12] and RacerPro[13]. Client interaction with reasoners differ depending on the reasoner used. For instance, RacerPro is designed for deployment on a server and interaction occurs over HTTP. Alternatively, Pellet allows client applications to interact with it directly using Java API method calls and also supports server deployment like RacerPro. Each reasoner is compatible with different XML/RDF Parsers and OWL Repositories, and contain their own Reasoner Loader components. Further details about the Pellet implementation will be given in Section 6.2, as we use this

[8]http://jena.sourceforge.net (accessed May 2009)
[9]http://owlapi.sourceforge.net (accessed May 2009)
[10]http://clarkparsia.com/pellet (accessed May 2009)
[11]http://kaon2.semanticweb.org (accessed May 2009)
[12]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[13]http://www.racer-systems.com (accessed May 2009)

open source reasoner for the implementation and evaluation of our proposed strategies.

As previously described in Section 2.2.1, the OWL language has three subsets: OWL-Lite, OWL-DL and OWL-Full. OWL-DL is based on Description Logic (DL) (Baader et al., 2005) and is the language subset which provides the "maximum level of expressiveness while retaining computational completeness and decidability, and can thus support accurate matching"(W3C, 2004). As a result, most of the commercial and widely used semantic reasoners support the OWL-DL language including Pellet, RacerPro and FaCT++. Thus, we focus on OWL-DL expressiveness in this thesis.

In addition, the World Wide Web Consortium (W3C) provides Semantic Web Services (SWS) also called OWL-S, which is specifically designed to describe Web Services (WS) (Martin et al., 2007). SWS is designed to facilitate matching of a request for functionality with Web Service offers. These allow a service provider or requester to describe a service or request using a profile. This profile allows the specification of a service description or request in terms of a set of service types, inputs, outputs, preconditions and effects. Service types, inputs and outputs are described using OWL class concept definitions while preconditions and effects are described using rule languages such as the Semantic Web Rule Language (SWRL)[14]. Our research is focused on OWL matching, and therefore, relates to matching service types, inputs and outputs. Rule languages are not the focus of our research.

In addition, while we focus on OWL matching, we do not restrict our approach to only matching of SWS profiles, which represent concrete services which can be invoked by remote requests over the Internet. While our approach will support this kind of matching our aim is to support a broader range of semantic matching. We wish to also support constraint based matching of

[14]http://www.w3.org/Submission/SWRL (accessed May 2009)

OWL-DL service descriptions beyond those that are invokable (e.g. a search for the location of a movie cinema).

In the next section we will provide an overview of the Description Logic language, which OWL-DL is based upon (Baader et al., 2005).

3.3 Description Logic Language

In the following we provide an overview of the formal definitions for Description Logic (DL) (Baader et al., 2003) since this is the logic language which OWL-DL is based upon (Baader et al., 2005), as described in the previous section. Furthermore, current Tableaux inference provers for the Semantic Web such as Pellet, RacerPro and FaCT++, use OWL-DL. DL is a decidable subset of First Order Predicate Logic (FOL), as described in Section 2.3. DL defines a knowledge base $\mathcal{K}$ which comprises a set of triples as shown in Equation 3.1.

$$\mathcal{K} = \langle \mathcal{C}, \mathcal{R}, \mathcal{X} \rangle \tag{3.1}$$

In Equation 3.1, $\mathcal{C}$ is a set of class definitions, $\mathcal{R}$ is a set of role[15] definitions and $\mathcal{X}$ is a set of individuals[16]. A knowledge base $\mathcal{K}$ has two parts, a TBox $\mathcal{T}$ which contains the terminological knowledge and an ABox $\mathcal{A}$ which contains assertional knowledge. The TBox contains all class definitions $\mathcal{C}$ and the role definitions $\mathcal{R}$. Classes define sets of individuals with common characteristics. Roles define the nature of relations between individuals. The ABox contains all individual assertions $\mathcal{X}$. An individual can be a member or instance of classes and can have connections to other individuals using the roles defined in the TBox. We provide an overview of formalised TBox knowledge (class and role definitions), ABox knowledge (individuals) and the semantics associated with this knowledge in the next subsections.

[15] A role can also be called a relation or property (Stuckenschmidt and Harmelen, 2002)

[16] An individual can also be called a node or an object (Stuckenschmidt and Harmelen, 2002)

3.3.1 Terminological Knowledge (TBox)

Classes describe common properties of real world objects. They can be organised into super and subclass hierarchies in order to support abstraction as defined in Definition 3.1, and they are contained in the TBox.

Definition 3.1. *Class concepts can be defined in the TBox $\mathcal{T}$ using the form: $C_1 \sqsubseteq C_2$ where C_1 and C_2 are class concepts, that could be read C_1 is a subclass of C_2.*

Definition 3.1 is known as a general concept inclusion (GCI) axiom (Baader et al., 2003, p. 18) and implies that C_1 contains a subset of the individuals contained in C_2. A GCI is transitive, meaning that $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_3$ implies $C_1 \sqsubseteq C_3$. Class equivalence is captured by GCI axioms, such that $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$ implies $C_1 \equiv C_2$, which is read as C_1 is equivalent to C_2. A formal definition will follow later in this section.

In order to define the formal semantics of class concepts, we consider an interpretation $\mathcal{I}$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns every atomic class C a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies TBox $\mathcal{T}$ iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for each $C_1 \sqsubseteq C_2 \in \mathcal{T}$ where C_1 and C_2 are class concepts. Such an interpretation is called a model $\mathcal{I}$ of $\mathcal{T}$. A concept C is satisfiable with respect to $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$ (Tsarkov and Horrocks, 2005). For instance, $\mathsf{Coffee} \sqsubseteq \mathsf{Beverage}$, implies that Coffee is interpreted to be a subset or equal to Beverage.

The interpretation function is extended by DL constructors. The DL constructors available differ depending on the level of expressiveness supported by a particular DL sub-language. For the purposes of this dissertation we consider the DL $\mathcal{SHOIN}$ language because this is used by the OWL-DL version 1.1 standard[17], which our proposed strategies are based upon. The $\mathcal{SHOIN}$

[17] http://www.w3.org/TR/2004/REC-owl-features-20040210 (accessed May 2009)

DL concept constructors (Baader and Sattler, 2001, p. 496) are defined in Table 3.1 where $\mathcal{I}$ denotes an interpretation. The concept constructors, in Table 3.1, include top[18], bottom[19], negation[20], conjunction[21], disjunction[22], universal quantification[23], existential quantification[24] and unquantifed number restriction which includes at-most[25], at-least[26] and exact[27] number restrictions.

Name	DL Syntax	Semantics
Top	$\top^{\mathcal{I}}$	$\Delta^{\mathcal{I}}$
Bottom	$\bot^{\mathcal{I}}$	$\emptyset$
Negation	$(\neg C)^{\mathcal{I}}$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
Conjunction	$(C_1 \sqcap C_2)^{\mathcal{I}}$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
Disjunction	$(C_1 \sqcup C_2)^{\mathcal{I}}$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
Univeral quant.	$(\forall R.C)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \mid \forall x_p.(x_r, x_p) \in R^{\mathcal{I}} \rightarrow x_p \in C^{\mathcal{I}}\}$
Existential quant.	$(\exists R.C)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \mid \exists x_p.(x_r, x_p) \in R^{\mathcal{I}} \wedge x_p \in C^{\mathcal{I}}\}$
At-most	$(\leq nR)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \mid \lvert\{x_p \in \Delta^{\mathcal{I}} \lvert (x_r, x_p) \in R^{\mathcal{I}}\}\rvert \leq n\}$
At-least	$(\geq nR)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \mid \lvert\{x_p \in \Delta^{\mathcal{I}} \lvert (x_r, x_p) \in R^{\mathcal{I}}\}\rvert \geq n\}$
Exact	$(= nR)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \mid \lvert\{x_p \in \Delta^{\mathcal{I}} \lvert (x_r, x_p) \in R^{\mathcal{I}}\}\rvert = n\}$

Table 3.1: Some Description Logic Concept Constructors

Since the constructors given in Table 3.1 give semantics to class concepts by creating interpretations $\mathcal{I}$, we can say two class concepts C_1 and C_2 are equivalent by writing $C_1 \equiv C_2$, if $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. For example, the concept $\forall$ sellsProduct.Coffee $\sqcap \forall$ sellsProduct.Beverage is equivalent to $\forall$ sellsProduct.(Coffee $\sqcap$ Beverage), which defines a coffee shop selling coffee, where coffee is a beverage. This is because applying the constructors to both concepts generates the same knowledge in $\mathcal{K}$.

[18]Top is called thing in OWL-DL, which means universally everything
[19]Bottom is called nothing in OWL-DL
[20]Negation is called complement in OWL-DL
[21]Conjunction is called intersection in OWL-DL
[22]Disjunction is called union in OWL-DL
[23]Universal quantifier is called all values from, in OWL-DL
[24]Existential quantifier is called some values from, in OWL-DL
[25]At-most number restriction is also called max cardinality in OWL-DL
[26]At-least number restriction is called min cardinality in OWL-DL
[27]Exact number restriction is called cardinality in OWL-DL

Roles are used in DL to form a structure between classes. Roles are binary, and can be defined by restricting their domain and range or as a sub-relation of another relation as defined in 3.2.

Definition 3.2. *Roles can be defined in the TBox $\mathcal{T}$ using one of the following forms:*

(a) $R \sqsubseteq C_1 \times C_2$ *where* C_1 *and* C_2 *are class concepts. This could be read as* R *has the domain* C_1 *and range* C_2*.*

(b) $R_1 \sqsubseteq R_2$ *where* R_1 *and* R_2 *are roles. This could be read as* R_1 *is a subset or equal to* R_2*.*

The Definition 3.2(a) specifies that if a role R connects an individual x_1 to an individual x_2 in the ABox $\mathcal{A}$, where the domain of R is class C_1 and the range of R is class C_2 in the TBox $\mathcal{T}$, then x_1 must be a member of C_1 and x_2 must be a member of C_2. The Definition 3.2(b) specifies that R_1 is a subrole of R_2 which implies that the set of connections such that x_1 connects to x_2 using the role R_1, is a subset of the connections such that x_1 connects to x_2 using the role R_2, where x_1 and x_2 are some individuals which belong to the ABox $\mathcal{A}$. For instance $\mathsf{sellsCoffee} \sqsubseteq \mathsf{sellsProduct}$ implies that the role sellsCoffee is interpreted to connect either a subset of or the same set of individuals that are connected by sellsProduct.

In order to define the formal semantics of roles, we consider an interpretation function which assigns to every role a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies TBox $\mathcal{T}$, with respect to roles iff $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ for each $R_1 \sqsubseteq R_2 \in \mathcal{T}$ where R_1 and R_2 are roles. If such an interpretation $\mathcal{I}$ can be found, with $R^{\mathcal{I}} \neq \emptyset$, then R is satisfiable with respect to $\mathcal{T}$.

The interpretation function for roles is extended by DL role constructors. OWL-DL[28] supports the role constructors (Baader and Sattler, 2001, p. 499) listed in Table 3.2.

[28] http://www.w3.org/TR/2004/REC-owl-features-20040210 (accessed May 2009)

Name	DL Syntax	Semantics
Inverse	$R^{-\mathcal{I}}$	$\{(x_p, x_r) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x_r, x_p) \in R^{-\mathcal{I}}\}$
Transitive	$R^{+\mathcal{I}}$	$\bigcup_{n\geq 1}(R^{+\mathcal{I}})^n$
Symmetric	$symm(R)^{\mathcal{I}}$	$\{(x_p, x_r) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x_r, x_p) \in R^{\mathcal{I}}\}$
Functional	$func(R)^{\mathcal{I}}$	$\{x_r \in \Delta^{\mathcal{I}} \Big\vert \lvert\{x_p \in \Delta^{\mathcal{I}} \lvert (x_r, x_p) \in R^{\mathcal{I}}\}\rvert \leq 1\}$
Inverse Functional	$invFunc(R)^{\mathcal{I}}$	$\{x_p \in \Delta^{\mathcal{I}} \Big\vert \lvert\{x_r \in \Delta^{\mathcal{I}} \lvert (x_r, x_p) \in R^{\mathcal{I}}\}\rvert \leq 1\}$

Table 3.2: Some Description Logic Role Constructors

The constructors given in Table 3.2 specify inverse roles R^-, transitive roles R^+, symmetric roles $symm(R)$, functional roles $func(R)$ and inverse functional roles $invFunc(R)$. For example, given $\mathsf{sellsProduct} \sqsubseteq \mathsf{Cafe} \times \mathsf{Coffee}$, in TBox $\mathcal{T}$, which implies the role sellsProduct has the domain Cafe and range Coffee, iff $\mathsf{sellsProduct}^- \equiv \mathsf{productSoldBy}$ then $\mathsf{productSoldBy} \sqsubseteq \mathsf{Coffee} \times \mathsf{Cafe}$ is true for $\mathcal{T}$.

An interpretation $\mathcal{I}$ which satisfies the TBox $\mathcal{T}$ for all concepts C and roles R in $\mathcal{T}$ is called a model $\mathcal{I}$ of $\mathcal{T}$. In other words, a model of $\mathcal{T}$ is an interpretation of $\mathcal{T}$ which does not contain any contradictions.

3.3.2 Assertional Knowledge (ABox)

Individuals are generally used to represent real world objects and are specified as instances of class concepts. An individual is defined by its membership to a class and by its relations to other individuals. Individuals are defined in Definition 3.3.

Definition 3.3. *Individuals can be defined in the ABox $\mathcal{A}$ using one of the following forms:*

(a) $C(x)$ where C is a class concept and x is an individual. This could be read as x is an instance of C.

(b) $R(x_1, x_2)$ where R is a role and where x_1 and x_2 are individuals. This could be read as x_1 is connected to x_2 by R.

The Definition 3.3(a), implies that the individual x is a member of the class C, in the ABox $\mathcal{A}$. The Definition 3.3(b) implies that individual x_1 is connected to individual x_2 by the role R, in the ABox $\mathcal{A}$. In addition, x_2 is said to be an R-neighbour of x_1. For example, if InstantCoffee and BobsCafe are individual names and Coffee is a class concept definition, then Coffee(InstantCoffee) means InstantCoffee is a type of Coffee and sells(BobsCafe, InstantCoffee) means that BobsCafe sells InstantCoffee. The ABox $\mathcal{A}$ is a finite set of such assertions. The TBox imposes semantic relationships between the concepts and roles used in the ABox.

ABoxes are given semantics by extending interpretations to individual names. An interpretation maps each individual name x to an element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. DL does not impose a unique name assumption, therefore, x_1 and x_2 are only considered distinct if specified such that $x_1 \neq x_2$ where x_1 and x_2 are individuals and an interpretation $\mathcal{I}$ satisfies $x_1 \neq x_2$ iff $x_1^{\mathcal{I}} \neq x_2^{\mathcal{I}}$. For instance, two individuals Coffee and Tea are not considered distinct unless specified as Coffee $\neq$ Tea.

The interpretation $\mathcal{I}$ satisfies the concept assertion $C(x)$ if $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and satisfies the role assertion $R(x_1, x_2)$, if $(x_1^{\mathcal{I}}, x_2^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation satisfies the ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$ if in addition to being a model of $\mathcal{A}$, it is a model of $\mathcal{T}$. Thus, a model of $\mathcal{A}$ and $\mathcal{T}$ is an abstraction of the concrete world where the following is valid:

- the concepts are interpreted as subsets of the domain as required by the TBox;
- the membership of the individuals to class concepts, and their relationships with one another in terms of roles, comply with the assertions of the ABox (Baader and Sattler, 2001, p. 64).

As such, in a model $\mathcal{I}$, concepts are interpreted as subsets of the domain Δ as required by the $\mathcal{T}$ and the membership of individuals to concepts and their relationships with one another, using roles, conform to the assertions in $\mathcal{A}$.

For example, suppose the TBox contains the class concept definition $\forall$ sellsProduct.Coffee and the ABox contains Coffee(InstantCoffee), and sellsProduct(BobsCafe, InstantCoffee) where BobsCafe and InstantCoffee are individuals. Then an interpretation $\mathcal{I}$ can be constructed which is a model of ABox $\mathcal{A}$ with respect to TBox $\mathcal{T}$, because InstantCoffee is an instance of the class Coffee. However, if the ABox contained the assertion $\neg$Coffee(InstantCoffee) rather than Coffee(InstantCoffee), meaning the InstantCoffee is an instance of the negation of Coffee, then an interpretation of ABox $\mathcal{A}$ cannot be constructed with respect to TBox $\mathcal{T}$. This is because $\forall$ sellsProduct.Coffee requires InstantCoffee to be an instance of Coffee, thus, creating a contradiction which invalidates the model.

Description Logic provides the basis to facilitate accurate matching of service requests with available services. However, matching requires the ability to check the validity of an inferred membership of an individual to a class. Such inference proof can be achieved using the Tableaux algorithm. As discussed previously Tableaux "has dominated recent DL research" (Baader et al., 2003, p. 322). This is primarily due to the fact that it can deal with highly expressive logics and reason with an arbitrary knowledge base, such as an ontology (Baader et al., 2003, p. 322). In addition, Tableaux incorporates many optimisation strategies which ensure efficient performance (Tsarkov et al., 2007), while retaining completeness, soundness, decidability and termination (Baader et al., 2003, p. 84). Furthermore, most current reasoners for the Semantic Web, such as FaCT++ (Tsarkov and Horrocks, 2006), Pellet (Sirin et al., 2007) and RacerPro[29], utilise the Tableaux algorithm. Therefore, our optimisation and adaptive reasoning strategies are built on the Tableaux algorithm. The next section provides the necessary background to readers about Tableaux.

[29]http://www.racer-systems.com/products/racerpro (accessed May 2009)

3.4 Tableaux

In our research, we propose and develop strategies which provide efficient and accurate matching of a service description against a user request. In order to support a high level of accuracy our approach involves performing semantic matching using an inference prover. In this section we introduce and provide an overview of the Tableaux algorithm which is an inference proof for Description Logic (DL). Our proposed mTableaux optimisation and caching strategies, which we will present in Chapter 4, apply to the Tableaux inference proof provided in this section.

3.4.1 Inference

The power of semantic representation means that explicit definitions and assertions can be used to infer new definitions and assertions using inference. Proving or disproving a conjectured inference, requires a decision proof. Truth tables were used as an early decision proof for propositional logic (O'Donnel et al., 2006, p. 109), which is a less expressive logic when compared to DL (Gore, 1998). However, truth tables are not feasible for logics containing a large number of propositional variables, and cannot solve more expressive logics such as DL. As a result, Tableaux was developed to provide a decision proof for more expressive logics with greater efficiency than truth tables. There are many different variants of Tableaux which support different logics (Gore, 1998), such as Modal Logics (Gore, 1998; Girle, 2001), Nonmonotonic Logics (Gore, 1998), Propositional Logic (Kelly, 1997), First Order Logic (Fitting, 1996) and Description Logic (Baader et al., 2003). Tableaux continues to be widely used as a DL decision proof.

Current Semantic Web reasoners use the OWL-DL language, which was discussed in Chapter 2 (see Section 2.2.1), since this is more expressive than

OWL-Lite, while also maintaining computational completeness and decidability (Baader et al., 2003, 84), which is not thought to be possible using OWL-Full (W3C, 2004). OWL-DL corresponds to DL, therefore, in order to reason with OWL-DL we focus on Tableaux for DL. OWL-DL is used to create an ontology which contains the set of logical constructs which directly correspond with the DL knowledge base.

Tableaux proves that semantic knowledge is satisfiable which means that there are no contradictions in the logic. Thus, Tableaux is known as an (un)satisfiability decision proof algorithm because inferences are proven by refuting the inference and proving this is not satisfiable. The negation of the conjectured inference is added to the knowledge base, and if after applying rules which implement the semantic interpretation $\mathcal{I}$ of the knowledge (which was described in Section 3.3), the knowledge base is unsatisfiable, then the inference is proven. Otherwise, it is disproven. As such, Tableaux reduces an inference check to be an (un)satisfiability problem.

Most current semantic reasoners use the Tableaux decision proof, including Pellet[30], FaCT++[31] and RacerPro[32]. Since inferences are reduced to an unsatisfiability problem, this means they must be performed on an consistent knowledge base, in order to be accurate. Therefore, before performing any inference checks, current reasoners perform a satisifiability check on the knowledge base, to ensure its consistency. If the knowledge base is consistent, these reasoners generally then check every possible inference, thus, making all implicit inferences explicit. The processes involved are known as classification (Baader et al., 2003, p. 72) which generates the complete TBox $\mathcal{T}$ and realisation (Baader et al., 2003, p. 47) which generates a complete ABox $\mathcal{A}$. The classification phase compares every pair of class concept definitions from the TBox to see if they have a subclass relationship. The realisation phase

[30]http://clarkparsia.com/pellet (accessed May 2009)
[31]http://owl.man.ac.uk/factplusplus (accessed May 2009)
[32]http://www.racer-systems.com (accessed May 2009)

performs an inference check between every individual in the ABox and every class in the TBox, to see if the individual is a member of the class.

In the next section we provide a formal overview of the Tableaux inference proof. We restrict our overview to the proving or disproving of a single conjectured inference, since this is the feature which we require a semantic reasoner to perform. Moreover we focus on the matching of a single service description against a user request in our approach as illustrated earlier in Figure 3.3 in Section 3.1.

3.4.2 Tableaux Inference Proof

Tableaux is a decision procedure which can provide inference proof for expressive logics such as DL while maintaining computational completeness and decidability, and provides reasonable performance. This is because it encompasses many optimisation strategies Tsarkov et al. (2007) which are employed by current semantic reasoners.

As stated in the previous section, the Tableaux decision procedure proves that a DL knowledge base is satisfiable and all inference checks are reduced to an (un)satisfiability problem for an ABox $\mathcal{A}$. ABox $\mathcal{A}$ inference checks can be about the relationship between class concepts, roles and individuals. Tableaux proves or disproves an inference by refuting it. Therefore, checking the validity of the inference $C(x)$, is reduced to the Equation 3.2.

$$\mathcal{A} \models C(x) \text{ iff } \mathcal{A} \cup \{\neg C(x)\} \text{ is inconsistent} \tag{3.2}$$

As discussed previously in Section 3.3, the semantics of DL definitions and assertions in the knowledge base are modelled as an interpretation $\mathcal{I}$. Interpretation $\mathcal{I}$ is mapped to definitions and assertions in the knowledge base using DL constructors. In order to create an interpretation, Tableaux has transformation rules which directly correspond to the DL constructors which we will discuss later in this section. Thus, an ABox $\mathcal{A}$ is expanded by application of

transformation rules to create ABox $\mathcal{A}'$, while conforming to the TBox $\mathcal{T}$. The expanded $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing each concept assertion $C(x)$ in $\mathcal{A}$ with the assertion $C'(x)$, where C' is the expansion of C in TBox $\mathcal{T}$. $\mathcal{A}$ is consistent with respect to $\mathcal{T}$ iff its expansion $\mathcal{A}'$ is consistent with respect to $\mathcal{T}$. Consistency is not achieved if an expansion into a complete interpretation $\mathcal{I}$ fails due to a logical contradiction. An ABox $\mathcal{A}$ contains a contradiction iff one of the following situations occur (Baader et al., 2003, p. 86), (Tsarkov et al., 2007, p. 282):

1. $\Big\{C(x), \neg C(x)\Big\} \subseteq \mathcal{A}$ for some individual name x and some concept name C. This means that the individual x is asserted to be a member of both a class concept C and its negation $\neg C$ in the ABox $\mathcal{A}$, thus creating a contradiction;

2. $\Bigg(\Big\{(\leq\ n\ R)(x_i)\Big\} \cup \Big\{R(x_i, x_j) \mid 1 \leq j \leq n+1\Big\}\ \cup\ \Big\{x_j \neq x_k \mid 1 \leq j \leq k \leq n+1\Big\}\Bigg) \subseteq \mathcal{A}$ for individual names x_i, x_j, x_k, and a role name R. This means that the individual x_i is specified to adhere to the definition $\{\leq\ \ n\ R\}$, where this definition implies that x_i must have no more than n connections to other individuals, by role R. However, x_i is in fact connected to at least $n+1$ number of distinct individuals x_j where $1 \leq j \leq n+1$, which contradicts $\leq\ n\ R$;

3. $\Big\{\bot(x)\Big\} \subseteq \mathcal{A}$ for some individual x. This means that x is asserted to the type nothing / bottom. For instance, suppose there is an individual x which is asserted to have the types C_i and C_j such that $C_i(x)$ and $C_j(x)$ and these concepts are disjoint such that $C_i \subseteq \neg C_j$. This would imply $\bot(x)$, due to a contradiction;

Therefore, an inference is proven if its negation is asserted to the ABox $\mathcal{A}$, and a complete expansion of $\mathcal{A}$ into $\mathcal{A}'$ contains a contradiction, meaning that a complete interpretation $\mathcal{I}$ of $\mathcal{A}$ cannot be constructed. Alternatively, if $\mathcal{A}$ is fully expanded into an $\mathcal{A}'$, where $\mathcal{A}'$ does not contain a contradiction, meaning

that a sound and non-contradictory interpretation $\mathcal{I}$ has been constructed, then the inference is disproven.

In practise, Tableaux expansion occurs by constructing a labelled completion graph, known as an expansion tree $\mathcal{G}$ (Baader et al., 2003, p. 323). This tree imposes ordering on the application of expansion rules. Successive ABox states resulting from expansions are maintained using labels. Two labels exist for the two kinds of ABox $\mathcal{A}$ assertions. The $\mathcal{A}$ assertion $C(x)$, where the individual x is a member of the class concept C, is denoted by $C \in \mathcal{L}(x)$, such that $\mathcal{L}(x)$ is a type label for x. The assertion $R(x, y)$ where R is a role connecting individuals x_1 and x_2, is denoted by $R \in \mathcal{L}(\langle x_1, x_2 \rangle)$ such that $\mathcal{L}$ is the edge label for connections between x_1 and x_2. The contents of labels depend on branch point nodes in the expansion tree $\mathcal{G}$, which we will discuss in Section 3.5 of this chapter.

Expansion occurs by application of Tableaux transformation rules which are applied until the model $\mathcal{I}$ is fully expanded. Tableaux transformation rules correspond to the DL constructors. We will now introduce each Tableaux transformation rule. The conjunction transformation rule is given in Figure 3.5.

$\sqcap$-rule: if 1. $C_1 \sqcap ... \sqcap C_n \in \mathcal{L}(x)$, and
2. $\{C_1, ..., C_n\} \nsubseteq \mathcal{L}(x)$
then $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{C_1, ..., C_n\}$

Figure 3.5: Conjunction Transformation Rule (Tsarkov et al., 2007, p. 283)

The conjunction rule is applied to an assertion where an individual x is asserted to be a member of a class concept definition $C_1 \sqcap ... \sqcap C_n$, $1 \leq i \leq n$ which is a conjunction. However, the rule is only applied when x is not already asserted to be a member of all of the conjunctive elements C_i. Application of the conjunction rule results in x being asserted as a member of each conjunctive element C_i.

The disjunction transformation rule is given in Figure 3.6.

$\sqcup$-rule: if 1. $C_1 \sqcup ... \sqcup C_n \in \mathcal{L}(x)$, and
2. $\{C_1, ..., C_n\} \cap \mathcal{L}(x) = \emptyset$
then $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, ..., C_n\}$

Figure 3.6: Disjunction Transformation Rule (Tsarkov et al., 2007, p. 283)

The disjunction rule is applied to an assertion where an individual x is asserted to be a member of a class concept definition $C_1 \sqcup ... \sqcup C_n$, $1 \leq i \leq n$, which is a disjunction. However, the rule is only applied when x is not asserted as a member of any of the disjunctive elements C_i. Application of the rule results in x being asserted as a member of one of the disjunctive elements C_i.

The existential quantifier transformation rule is given in Figure 3.7.

$\exists$-rule: if 1. $\exists R.C \in \mathcal{L}(x_i)$, and
2. there is no R-neighbour x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $C \in \mathcal{L}(x_j)$
then create a new node x_j with $\mathcal{L}(\langle x_i, x_j \rangle) = \{R\}$ and $\mathcal{L}(x_j) = \{C\}$

Figure 3.7: Existential Quantifier Transformation Rule (Tsarkov et al., 2007, p. 283)

The existential quantifier transformation rule is applied when an individual x_i is asserted to be a member of an existential quantifier definition of the form $\exists R.C$. However, it is only applied if x_i does not have an R-neighbour x_j, where x_j is asserted to be a member of the class concept C. An R-neighbour means that x_i is connected to the individual x_j by the role R. When the $\exists$-rule is applied it creates a new R-neighbour x_j, for x_i, and asserts that x_j is a member of C.

The universal quantifier transformation rule is given in Figure 3.8.

The universal quantifier transformation rule is applied to an assertion where an individual x_i is asserted to be a member of a class concept definition $\forall R.C$, which is a universal quantifier. However, the rule is only applied when x_i has at

∀-rule: if
1. $\forall R.C \in \mathcal{L}(x_i)$, and
2. there is some R-neighbour x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $C \notin \mathcal{L}(x_j)$

then $\mathcal{L}(x_j) \leftarrow \mathcal{L}(x_j) \cup \{C\}$

Figure 3.8: Universal Quantifier Transformation Rule (Tsarkov et al., 2007, p. 283)

least one R-neighbour x_j, where x_j is not asserted to be a member of the class concept C and an R-neighbour implies that x_i is connected to the individual x_j by the role R. Application of the ∀-rule results in all R-neighbours x_j, of x_i being asserted as members of the class concept C.

The minimum cardinality transformation rule is given in Figure 3.9.

≥-rule: if
1. $\geq\ n\ R \in \mathcal{L}(x_i)$, and
2. there are no R-neighbours x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $x_j \neq x_k$, $1 \leq j \leq k \leq n$

then create new nodes x_j with $\mathcal{L}(\langle x_i, x_j \rangle) = \{R\}$, and $x_j \neq x_k$, $1 \leq j \leq k \leq m$, so that $m = n$

Figure 3.9: Minimum Cardinality Transformation Rules (Tsarkov et al., 2007, p. 283)

The minimum cardinality transformation rule is applied to an assertion where an individual x_i is asserted to be a member of a class concept definition $\geq\ n\ R$. This definition requires that x_i must have at least n number of distinct R-neighbours x_j. An R-neighbour of x_i implies that x_i connects to another individual x_j using the role R. Therefore, the ≥-rule is fired / applied if x_i does not have n number of R-neighbours where no two R-neighbours x_j and x_k, $1 \leq i \leq j \leq k$, are the same. When the rule is applied it creates additional distinct R-neighbour individuals for x_i, to meet the required n number of R-neighbours.

The maximum cardinality transformation rule is given in Figure 3.10.

The maximum cardinality transformation rule is applied to an assertion where an individual x_i is asserted to be a member of a class concept definition

$\leq$-rule: if
1. $\leq\ n\ R \in \mathcal{L}(x_i)$, and
2. there are R-neighbours x_j of x_i, such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$, $1 \leq j \leq m$, where $m > n$ and there is a pair of R-neighbours which is not declared $x_j \neq x_k$, $1 \leq j \leq k \leq m$

then for some pair x_j, x_k which is not declared $x_j \neq x_k$, $Merge(x_j, x_k)$ for $1 \leq j \leq k \leq m$, where,

$Merge(x_j, x_k)$:
1. for all neighbours x_p of x_j, such that $\mathcal{L}(\langle x_j, x_p \rangle)$:
   (a) set $\mathcal{L}(\langle x_k, x_p \rangle) \leftarrow \mathcal{L}(\langle x_k, x_p \rangle) \cup \mathcal{L}(\langle x_j, x_p \rangle)$;
   (b) set $\mathcal{L}(\langle x_p, x_k \rangle) \leftarrow \mathcal{L}(\langle x_p, x_k \rangle) \cup \{R_s^-\}$ iff $R_s \in \mathcal{L}(\langle x_j, x_p \rangle)$;
   (c) set $\mathcal{L}(\langle x_j, x_p \rangle) \leftarrow \emptyset$
2. for all neighbours x_j of x_p, such that $\mathcal{L}(\langle x_p, x_j \rangle)$:
   (a) set $\mathcal{L}(\langle x_p, x_k \rangle) \leftarrow \mathcal{L}(\langle x_p, x_k \rangle) \cup \mathcal{L}(\langle x_p, x_j \rangle)$;
   (b) set $\mathcal{L}(\langle x_k, x_p \rangle) \leftarrow \mathcal{L}(\langle x_k, x_p \rangle) \cup \{R_s^-\}$ iff $R_s \in \mathcal{L}(\langle x_p, x_j \rangle)$;
   (c) set $\mathcal{L}(\langle x_p, x_j \rangle) \leftarrow \emptyset$
3. set $\mathcal{L}(x_k) \leftarrow \mathcal{L}(x_k) \cup \mathcal{L}(x_j)$;
4. set $\mathcal{L}(x_j) \leftarrow \emptyset$;
5. set $x_k \neq x_y$ for all individuals x_y where $x_j \neq x_y$

Figure 3.10: Maximum Cardinality Transformation Rule (Tsarkov et al., 2007, p. 283)

$\leq\ n\ R$. This class concept defines that x_i cannot connect to more than n number of individuals via role R. Therefore, the $\leq$-rule will be applied / fired if x_i has more than n number of R-neighbours, where at least one pair of R-neighbours are not declared as distinct (i.e. distinct individuals cannot be merged / combined). An R-neighbour of x_i is an individual x_j which x_i connects to using the role R. If the conditions for the $\leq$-rule are met then the rule attempts to merge each pair x_j and x_k of R-neighbours from x_i, into a single individual, where x_j and x_k are not declared as being distinct, in order to meet the maximum cardinality requirement. This involves merging the individual x_j into x_k as follows. Any role R_s relations which connect x_j to some other individual x_p, are changed to connect x_k to x_p instead, and the inverse role R_s^- is set to connect x_p to x_k. Inverse roles were discussed in Section 3.3. Additionally, any role R_s relations which connect some individual x_p to x_j, are changed to connect x_p to x_k instead and the inverse R_s^- is set to

connect x_k to x_p. The type label contents for the individual x_j is also moved to the type label for x_k. Finally, if there are any declarations which set x_j to be distinct from some other individual x_y then x_k is set to be distinct from x_y.

Usually there will be many transformation rules which are simultaneously applicable. Most current implementations of Tableaux maintain a *ToDo* list of assertions which Tableaux transformation rules are applicable to (Tsarkov et al., 2007, p. 293), in order to avoid checking all assertions for this applicability. If a transformation is applicable to a particular assertion, by meeting the conditions of the rule (which were defined earlier in this section), then the assertion is added to a *ToDo* list, meaning the rule is ready to be fired / applied. *ToDo* lists are employed by current reasoners and group assertions by transformation rule type. For instance, the $\forall$-rule may be applied to all applicable $\forall R.C$ assertions first, then the $\exists$-rule is applied to all applicable $\exists R.C$ terms, etc.

The order in which transformation rules are applied does not affect correctness, however, it can have a significant impact on efficiency (Tsarkov and Horrocks, 2005). Different reasoners give different priorities to transformation rule types, in terms of the order in which they are applied. Empirical analysis shows the $\sqcup$-rule and $\exists$-rule should be given the lowest priority, because these are the most costly rules (Tsarkov and Horrocks, 2005) in terms of performance. The $\sqcup$-rule increases the size of the search space because each of its disjunct elements represent alternative expansions to explore (which we will describe in more detail, later in this section). The $\exists$-rule increases the size of the search space by generating new individuals. The performance impact for the application order of the $\sqcup$-rule and the $\exists$-rule also depends on the knowledge in the ABox $\mathcal{A}$ and TBox $\mathcal{T}$ (obtained from an ontology) over which the reasoning is being performed. In addition, some reasoners, such as Pellet[33], apply the $\sqcap$-rule and the $\forall$-rule on a class concept C as soon as C is added to

[33]http://clarkparsia.com/pellet (accessed May 2009)

$\mathcal{L}(x)$ where x is an individual in $\mathcal{A}$, rather than adding these to the $ToDo$ list (Tsarkov et al., 2007, p. 298).

In this section, so far we have discussed the Tableaux transformation rules which are designed alter the state of an ABox in order to model the semantics implied by the Description Logic constructs. However, some transformation rules can generate multiple possible alternative ABox states. The $\sqcap$-rule, $\exists$-rule, $\forall$-rule and $\geq$-rule are said to be deterministic rules because these rules deterministically transform the ABox $\mathcal{A}$ to another single ABox $\mathcal{A}'$ state. The $\exists$-rule and $\geq$-rule are also said to be generator rules, because these cause new individuals to be created in $\mathcal{A}$. However, the $\sqcup$-rule and $\leq$-rule are said to be non-deterministic because they expand the ABox $\mathcal{A}$ into a finite number of ABoxes $\mathcal{A}_1$, ..., $\mathcal{A}_m$ but only one of the new ABoxes needs to be consistent in order for the original ABox to be consistent. Therefore, Tableaux must expand each ABox $\mathcal{A}_1$, ..., $\mathcal{A}_m$, until either a consistent one is found or all have been expanded. A mechanism is required which controls expansion of non-deterministic transformation rules because each finite expansion state may need to be explored in order to find a consistent ABox. This control mechanism is known as an expansion tree $\mathcal{G}$. We will provide an overview of the expansion tree in the remainder of this section. Then in Section 3.5 we will discuss this mechanism in greater detail.

The Tableaux expansion tree $\mathcal{G}$ is used to guide the search through possible ABox expansions which can be generated by non-deterministic transformation rules, until a consistent expansion can be found. The expansion tree is considered to be fully expanded when no more expansion rules can be applied. Every assertion in a label $\mathcal{L}$ or in the $ToDo$ list of transformation rules to apply, is associated with a branch point node b in the expansion tree $\mathcal{G}$, to which the assertion is said to depend upon. This dependency is defined as follows (the first two points relate to the labels $\mathcal{L}$ and the last point relates to the $ToDo$ list):

1. A concept $C_1 \in \mathcal{L}(x)$ depends on a branching point b_i if C_1 was added to $\mathcal{L}(x)$ at that branching point b_i or if C_1 depends on another concept C_2, and C_2 depends on the branching point b_i. A concept $C_1 \in \mathcal{L}(x)$ depends on a concept C_2 when C_1 was added to $\mathcal{L}(x)$ by the application of a deterministic transformation rule that used C_2;

2. A role $R \in \mathcal{L}(\langle x_1, x_2 \rangle)$ depends on a branching point b_i if R was added to $\mathcal{L}(\langle x_1, x_2 \rangle)$ at that branching point b_i or if R depends on concept C_2, and C_2 depends on the branching point b_i. A role $R \in \mathcal{L}(\langle x_1, x_2 \rangle)$ depends on a concept C_2 when R was added to $\mathcal{L}(\langle x_1, x_2 \rangle)$ by the application of a deterministic expansion rule that used C_2;

3. An assertion $C_1(x)$ in the $ToDo$ list depends on the branch point b_i or concept C_2 which $C_1 \in \mathcal{L}(x)$ depends on.

For example if C was asserted as a type to x_2 implying $C \in \mathcal{L}(x_2)$, due to the expansion of $\forall R.C \in \mathcal{L}(x_1)$, then $C \in \mathcal{L}(x_2)$ depends on $\forall R.C \in \mathcal{L}(x_1)$, where C is a class concept, R is a role relation and x_1, x_2 are individuals.

When a contradiction occurs, which is caused by the existence of the assertion $C \in \mathcal{L}(x)$ or a $R \in \mathcal{L}(\langle x_1, x_2 \rangle)$, then the ABox $\mathcal{A}$ is restored to its earlier state at the branch point b_i, where the assertion causing the clash depends on b_i or another concept C_2 where C_2 depends on b_i. Then another branch can be explored to try and alleviate the cause of the clash.

For instance, let b_i and b_{i+1} be two branch points, where b_i is an ancestor of b_{i+1}. Thus, the branch point b_i appears at a level less than b_{i+1} in the expansion tree $\mathcal{G}$. Assume adding C to the label $\mathcal{L}(x_1)$, generates a clash, because the label already contains the negation of C (i.e. $\neg C \in \mathcal{L}(x_1)$). In addition, assume C depends on b_i, meaning that the ABox $\mathcal{A}$ is restored to branch b_i. Restoring to branch point b_i means that any assertion which depends on branch point b_{i+1}, is removed, where an assertion may be a class concept C in the label $\mathcal{L}(x_1)$; or role R in the label $\mathcal{L}(\langle x_1, x_2 \rangle)$; or assertion $C(x)$ in the $ToDo$ list,

for all individuals x_1 and x_2 in the ABox $\mathcal{A}$. In addition, any assertion which was added to a label $\mathcal{L}$ or $ToDo$ list before b_i, but was removed after b_i, must be added back to the label or $ToDo$ list when the ABox $\mathcal{A}$ restored to b_i. For instance, application of the $\leq$-rule may remove an edge R from an edge label $\mathcal{L}(\langle x_1, x_2 \rangle)$, due to the merger of x_2 with another individual. If this merger occurs at branch b_{i+1} and the ABox is being restored to an earlier branch b_i, then the edge R should be added back to $\mathcal{L}(\langle x_1, x_2 \rangle)$.

The Tableaux decision procedure has been highly optimised using various strategies (Tsarkov et al., 2007) which are generally incorporated as standard in the current semantic reasoners, such as FaCT++, Pellet and RacerPro. Tableaux also ensures that the desirable properties of soundness, completeness, consistency, termination and decidability are upheld (Baader et al., 2003, p. 85). However, despite its efficiency, Tableaux remains a resource intensive operation which challenges mobile reasoning. In the next section we provide an example to illustrate the way in which Tableaux proves an inference, by applying transformation rules. This example will show that some rules contribute to the inference proof, while others do not, which motivates our proposed optimisations to enable Tableaux to function on mobile devices. This will form the basis for our mTableaux optimisation strategies, which are presented in Chapter 4 in order to enable mobile inference checks.

3.4.3 Example Inference Proof

We will now illustrate Tableaux using an example. Let there be a consistent ABox $\mathcal{A}$, where $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\} \subseteq \mathcal{A}$ and x_i are individuals. The individuals are explicitly asserted to be members of class concept definitions, such that, $\mathcal{L}(x_3) = \{C_1, \neg C_4, C_4 \sqcup C_5\}$, $\mathcal{L}(x_4) = \{C_2\}$, $\mathcal{L}(x_5) = \{C_2, C_3\}$, $\mathcal{L}(x_6) = \{\forall R_3.(\neg C_1 \sqcup \neg C_2\}$, $\mathcal{L}(x_7) = \{C_1\}$. This is illustrated in Figure 3.11. Also suppose that the individuals also have the role relations which

are illustrated in the figure, such that an individuals x_i connects to another individual x_j using the role R_p.

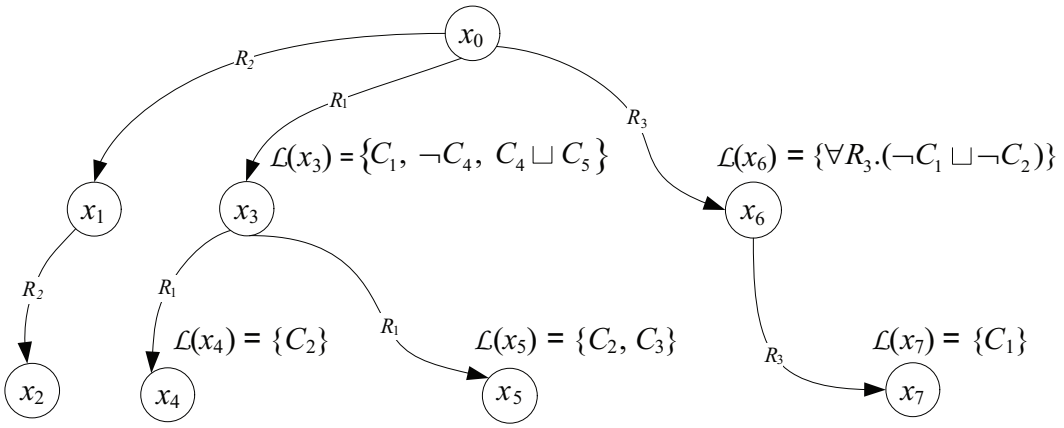


Figure 3.11: Tableaux Proof Example: ABox

Suppose we wish to check the validity whether the individual x_0 is a member of the class concept definition C_0, such that $C_0(x_0)$, where $C_0 \equiv \exists R_2.(\geq 1 R_2) \sqcap \exists R_1.(C_1 \sqcap \exists R_1.(C_2 \sqcap C_3))$. For instance, the first part of the definition C_0 implies that these exists an individual x_0 which connects to another individual x_r using rule R_2 and x_r connects to a maximum of one individual using role R_2. In order to perform this inference check, Tableaux asserts the negation of the inference to the ABox $\mathcal{A}$, which involves adding $\neg C_0$ to the type label of the individual x_0, such that $\mathcal{L}(x_0) \leftarrow \mathcal{L}(x_0) \cup \{\neg C_0\}$. The negated definition of C_0 is $\neg C_0 \equiv \forall R_2.(\leq \emptyset R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$. The ABox $\mathcal{A}$ is then expanded into ABoxes $\mathcal{A}_1$, ..., $\mathcal{A}_m$, $1 \leq j \leq m$, by applying transformation rules. If no clash free expansion $\mathcal{A}_j$ can be found, then the inference is proven. The application of transformation rules to check the validity of $C_0(x_0)$, is illustrated in Figure 3.12. Note, in the figure, the transformation rule and the assertion it is applied to, are highlighted in blue. The transformation / action which was completed by the rule is highlighted in purple. If the rule application lead to a clash, this is highlighted in red. The contradiction is given to the right of the red clash highlight.

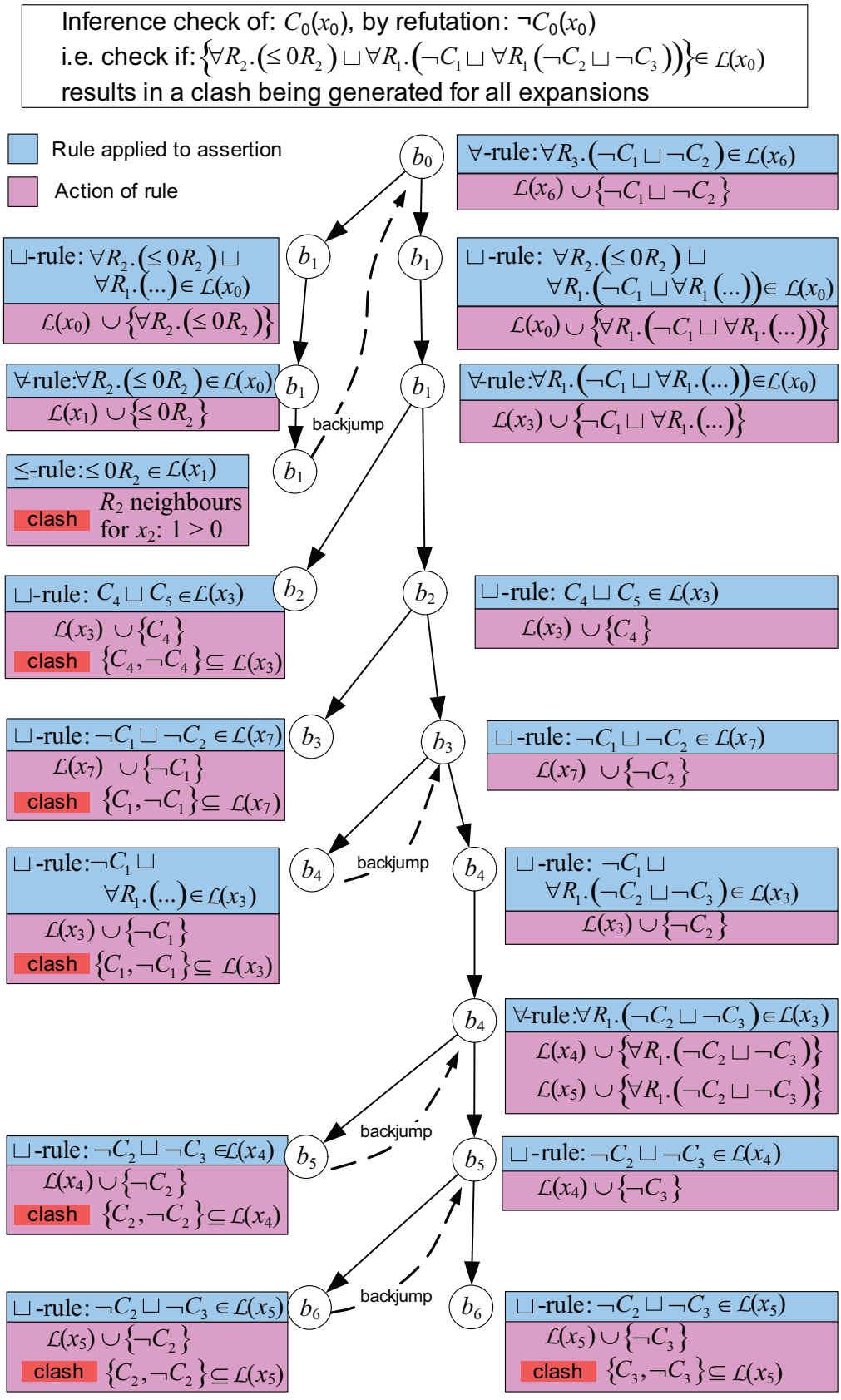


Figure 3.12: Tableaux Proof Example: Tableaux Transformation Rule Expansion

In this example, the inference process begins at branch node b_0. The action of adding $\neg C_0$ as a type for the individual x_0, in order to perform the inference check $C_0(x_0)$ makes the $\sqcup$-rule applicable to this assertion, because the definition $\neg C_0$ is a disjunction. Therefore, after the $\forall$-rule is applied to an assertion which was already in the ABox, $\sqcup$-rule is applied to $\neg C_0 \in \mathcal{L}(x_0)$ at branch node b_0, resulting in the creation of a new branch point node b_1 which is added to the expansion tree $\mathcal{G}$. This makes b_1 (left side in the figure) the currently active branch node. Then, the $\sqcup$-rule asserts the first disjunct element member, which is $\forall R_2.(\leq \emptyset R_2)$, as a type for the individual x_0, by adding it to the type label, such that $\mathcal{L}(x_0) \leftarrow \mathcal{L}(x_0) \cup \{\forall R_2.(\leq \emptyset R_2)\}$. The element member is a universal quantifier definition. Therefore the $\forall$-rule is then applied to it. The $\forall$-rule asserts the universal quantifier's role filler class concept definition $\leq \emptyset R_2$ to all individuals which x_0 connects to using the role R_2. x_0 connects to one such individual x_1 using the role R_2 (see Figure 3.11). Therefore $\leq \emptyset R_2$ is asserted as a type for x_1, such that $\mathcal{L}(x_1) \leftarrow \mathcal{L}(x_1) \cup \{\leq \emptyset R_2\}$. This definition $\leq \emptyset R_2$ is a maximum cardinality restriction which the $\leq$-rule is applicable to. The definition requires that x_1 is connected to zero individuals using the role R_2. However, x_1 is in fact connected to the individual x_2 using the role R_2, which violates the requirement causing the $\leq$-rule to detect a clash. This clash means that the attempted ABox $\mathcal{A}$ expansion has failed and a backjump is required to try an alternative branch.

A backjump will restore the reasoner to branch point b_1 because $\leq \emptyset R_2 \in \mathcal{L}(x_1)$ which caused the clash depends on branch b_1 as it was added when b_1 was the active branch point node. Note restoring to branch node b_1, implies that all assertions which were added at or after b_1, are discarded (e.g. $\forall R_2.(\leq R_2)$ depends on b_1 and is thus discarded). The branch b_1 was created by the $\sqcup$-rule for the disjunction $\forall R_2.(\leq \emptyset R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$ which is a type for x_1. This disjunction has two disjunct elements, and only the first

has so far been evaluated. Therefore, the second disjunct element, which is $\forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$, is then added as a type for x_1.

This process continues, until all ABox expansion possibilities have generated a clash. This is proven if a disjunction which depends on branch point node b_0 in the expansion tree $\mathcal{G}$, has generated a clash for all of its disjunctive elements. In Figure 3.12, the application of the $\sqcup$-rule on the disjunction $\neg C_2 \sqcup \neg C_3 \in \mathcal{L}(x_5)$ generated a branch node b_6, and clashed for both expansions. Branch node b_6 depends on the second expansion (right side in the figure) of branch node b_4 because the disjunction $\neg C_2 \sqcup \neg C_3$ was added as a type for the individual x_3 at the second expansion (right side in the figure) of b_4. Therefore, the second expansion of b_4 has generated a clash. The first expansion of b_4 also generated a clash due to a contradiction of the concept $\neg C_1$ for the individual x_3. The branch node b_4, thus, clashes for all possible expansions. The node b_4 depends on the second expansion of b_1 because it was generated by the application of $\neg C_2 \sqcup \neg C_3$ which was added as a type to x_0 at the second expansion of b_1. Thus the second expansion of b_1 generates a clash. The first expansion of b_1 also generated a clash due to the class concept $\leq \emptyset R_2$ added to the individual x_1. The node b_1 now clashes for both expansions. The node b_1 depends on b_0 because it was added when b_0 was the active branch node. This proves that b_0 and all possible expansions which depend on b_0 have generated a clash. Therefore, all attempts to construct a consistent model have led to a contradiction, where the negation of C_0 is added as a type for the individual x_0. This proves that x_0 can be inferred to be a member of the class concept C_0.

In this example, it can be seen that many transformation rules were applied which did not contribute to the clashes which proved the inference. In Figure 3.13 the transformations / expansions which contributed to the proof of the inference are highlighted in yellow and circled. The reason we know this, is because in order to prove the inference we assert its negation and attempt

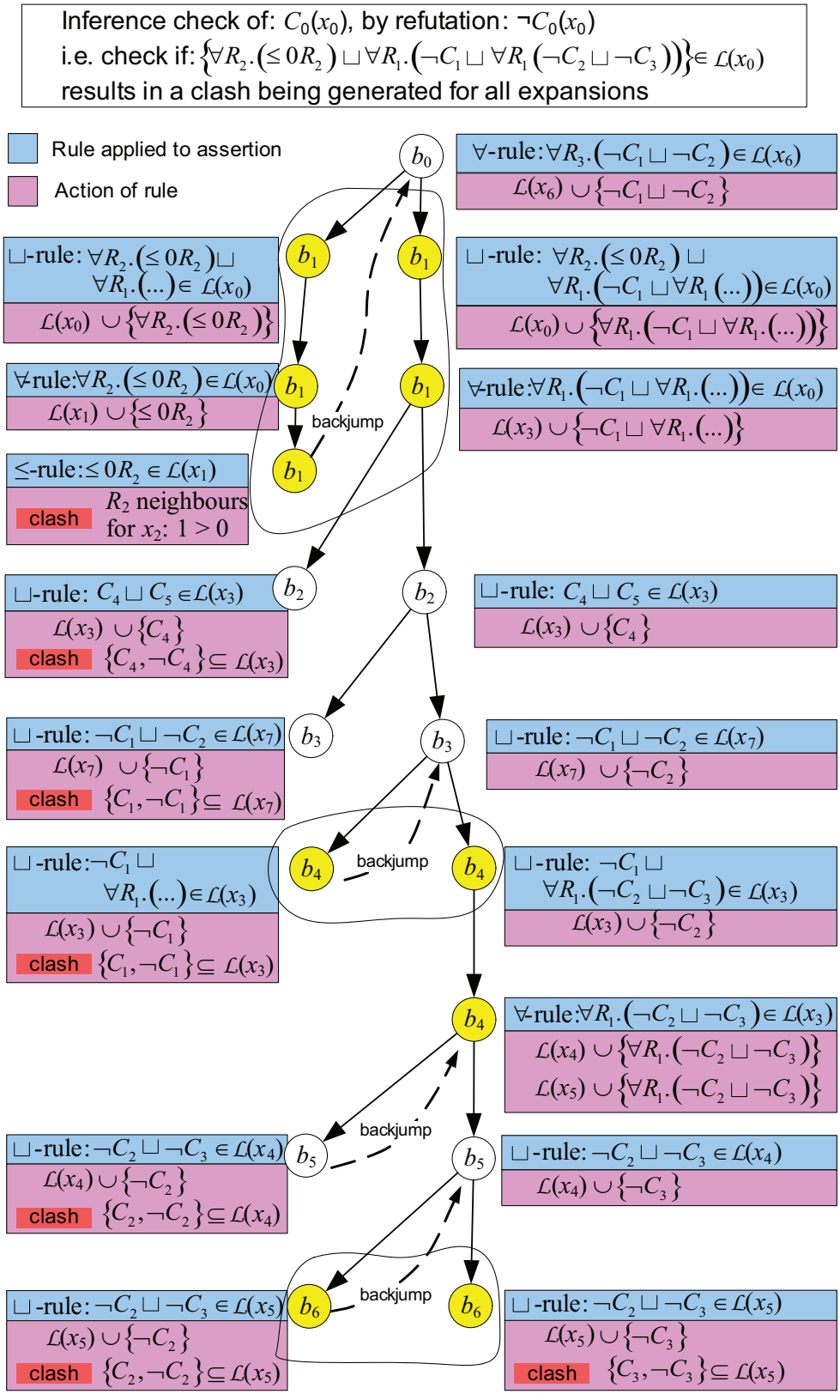


Figure 3.13: Tableaux Proof Example: Highlighting the transformations which were necessary to prove the inference

to prove a clash exists for every possible expansion. As such we asserted $\forall R_2.(\leq\ \emptyset R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$ as a type for x_0. The first disjunction element was proven to clash by applying the $\leq$-rule at branch point node b_1 to $\leq \emptyset R_2$. Therefore, $\leq \emptyset R_2$ and all assertions which it depends on contributed to the clash. The second disjunct element of the negated inference was proven when the last disjunction at branch node b_6 in Figure 3.13 clashed for both disjunct elements. Therefore, this disjunction $\neg C_2 \sqcup \neg C_3$, and all assertions which it depends on, contributed to the clash. Note, the disjunctions $C_4 \sqcup C_5 \in \mathcal{L}(x_3)$ and $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_7)$, generated a clash for the first expansion, but did not contribute to the inference proof because they did not generate a clash for the second expansion.

Applying the $\forall$-rule on $\forall R_3.(\neg C_1 \sqcup \neg C_2) \in \mathcal{L}(x_6)$, the $\sqcup$-rule on $C_4 \sqcup C_5 \in \mathcal{L}(x_3)$, $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_7)$ and $\neg C_2 \sqcup \neg C_3 \in \mathcal{L}(x_4)$, and any transformations which resulted from these transformations, did not assist in proving the inference. In more realistic examples, disjunction element members would often contain other definitions which would result in the subsequent application of transformation rules to these definitions. For instance, suppose there is a class concept definition $C_4 \equiv \exists R_1.C_6$ and $C_4 \sqcup C_5$. The application of $C_4 \sqcup C_5$ would result in the additional application of the $\exists$-rule on C_4. In addition, in realistic scenarios, the ABox which is filled by assertions from an ontology, would generally contain a much greater number of individuals which have definitions asserted as their types. In this case, there would be a much greater number of transformation rules applied, which may not be useful in proving the inference. Therefore, in this thesis, we propose that the elimination of these rule applications / inference checks would considerably improve efficiency without reducing accuracy. As a significant contribution of our research, we propose and develop mTableaux which incorporates strategies which reduce the application of transformation rules to assertions which do not aid in the inference proof, without compromising the validity of the inference proof.

In this section we have provided an overview of the Tableaux algorithm and all of its transformation rules. In the next section we will provide further explanation of the process of expansion using the Tableaux expansion tree $\mathcal{G}$. This explanation will be required in order to understand Chapter 5, which alters the order in which Tableaux expansion occurs, to enable adaptive reasoning to provide intermediate results from inferencing processes which are interrupted due to limits of time, resources, or other user constraints.

3.5 Tableaux Expansion Search Tree

As discussed in Baader et al. (2003, p. 313) and Cormen et al. (2001, p. 540) the Tableaux algorithm performs expansion in depth-first order. Therefore, in the next section will provide a more detailed discussion of the depth-first expansion which is used to control the expansion process of current Tableaux reasoners using search tree branching.

3.5.1 Tableaux Depth-First Expansion

As described in Section 3.4.2, Tableaux applies transformation rules which expand the ABox $\mathcal{A}$ according to the semantics imposed by the DL language constructors (see Section 3.3). More specifically Tableaux imposes an ordering on the expansions generated by subsequent application of transformation rules, using an expansion tree $\mathcal{G}$. This tree contains branch point nodes b. The application of $\sqcup$-rule to a disjunction, gives rise to a new branch point node in the tree and the tree is expanded in depth-first order. Expansion occurs either until no more expansions are possible or a clash is detected causing a backjump so that an alternative earlier branch can be explored. Using this tree expansion, Tableaux proves an inference by asserting its negation, and if a fully expanded tree cannot be constructed due to clashes, then the inference

is proven, otherwise it is disproven. This depth first tree expansion process is illustrated in algorithmic form in Algorithm 3.1.

Algorithm 3.1 TableauxTreeTraverse($\mathcal{A}$, $\mathcal{G}$)

Inputs: ABox $\mathcal{A}$, ExpansionTree $\mathcal{G}$
Outputs: Boolean $allExpansionsClash$

```
1: Let ClassConcept clash ← null
2: while moreTableauxRulesToApply(A, G) = true do
3:    clash ← ApplyTableauxRules(A, G) \*standard Tableaux rules*\
4:    if clash ≠ null then
5:       Let openBranchFound ← BackJump(clash, A, G)
6:       if openBranchFound = false then
7:          return true
8:       end if
9:    end if
10: end while
11: return false
```

The algorithm continues applying transformation rules until no more rules can be applied, by looping on the *moreTableauxRulesToApply* algorithm. Let *moreTableauxRulesToApply* be an algorithm (which is not shown) that returns *true* if there are more transformation rules to apply, otherwise it returns $false$. As we discussed in Section 3.4.2, an assertion which has a transformation rule applicable to it is stored in the $ToDo$ list. Therefore, *moreTableauxRulesToApply* returns *true* while the $ToDo$ list is not empty. The *ApplyTableauxRules*[34] algorithm applies all applicable transformation rules to the ABox $\mathcal{A}$. The *ApplyTableauxRules* algorithm will apply the appropriate transformation rules to the assertions in the $ToDo$ list and remove them from this list. The application of a transformation rule to $\mathcal{A}$ may result in a contradiction (clash), in which case the class concept which caused the clash is returned by *ApplyTableauxRules*. When a clash occurs at branch b_{i+1}, the reasoner jumps back to an earlier branch point b_i in $\mathcal{G}$ and explores an alternative branch which might alleviate the cause of the clash. Backjumping is a standard Tableaux optimisation (Tsarkov et al., 2007, p. 294)

[34]The *ApplyTableauxRules* function applies the standard Tableaux transformation rules as outlined in Section 3.4.2 and is therefore not shown

which is employed by current reasoners. Backjumping is carried out by the *BackJump* algorithm which we will describe later in this section (see algorithm 3.3). *BackJump* returns *true* if an alternative branch has been found, in which case $TableauxTreeTraverse$ continues applying more transformation rules. This process repeats until either every branch alternative generates a clash or no more transformation rules can be applied to the current branch.

In Algorithm 3.2 we show an algorithmic implementation of the application of the $\sqcup$-rule transformation, which creates a new branch point node.

Algorithm 3.2 ApplyDisjRule(C, x, $\mathcal{A}$, $\mathcal{G}$)

Inputs: ClassConcept C, Individual x, ABox $\mathcal{A}$, ExpansionTree $\mathcal{G}$
Outputs: ClassConcept *clash*
Pre-conditions: C is a disjunction added as a type to x, such that $C \in \mathcal{L}(x)$, $C \equiv (E_1 \sqcup ... \sqcup E_m)$ and $x \in \mathcal{A}$

1: Let b_j be newly created branch node
2: Let b_i be the last added branch node to $\mathcal{G}$ (highest $branchID(b_i)$ for all $b_i \in \mathcal{G}$)
3: $branchID(b_j) \leftarrow branchID(b_i) + 1$
4: $\mathcal{G} \leftarrow \mathcal{G} \cup \{b_j\}$
5: $createdByAssertion(b_j) \leftarrow C(x)$
6: $depOnBranch(b_j) \leftarrow depOnBranch(C)$
7: $depOnBranch(E_1) \leftarrow b_j$
8: $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{E_1\}$ *adds E_1 as a type for x*\
9: **if** $\{E_1, \neg E_1\} \subseteq \mathcal{L}(x)$ **then**
10: **return** E_1 *immediate clash*\
11: **end if**
12: **return null**

In the algorithm, $\sqcup$-rule, being applied to the next disjunction class concept definition C which is a type for the individual x, from the *ToDo* list of applicable $\sqcup$-rules. The algorithm creates a new branch point node b_j which is added to $\mathcal{G}$ and adds one of the disjunct elements of the disjunctive concept C as a type for the individual x, which implements the functionality of the $\sqcup$-rule (see Section 3.4.2). Each new b_j is given an identifier by the algorithm, which an ascending depth count integer. Let $branchID(b_j)$ denote this identifier for the branch point node b_j. Each branch point node, is also associated with the assertion which, after having the $\sqcup$-rule to it, resulted in the branch node's

creation. Let $createdByAssertion(b_j)$ denote the type assertion $C(x)$ which the $\sqcup$-rule was applied to, which resulted in the creation of a new branch b_j. The algorithm 3.2 then adds the first disjunct element E_1 from the disjunctive concept C as a type for individual x. As discussed previously in Section 3.4.2, whenever an assertion is added to or removed from the ABox $\mathcal{A}$ or $ToDo$ list, this action is said to depend on a branch point node. An assertion is said to depend on b_j if it was added to a type or role label $\mathcal{L}$ or the $ToDo$ list, at branch point b_j or by the application of a deterministic transformation rule on a class concept C where C depends on b_j. Let $depOnBranch(C)$ denote the b_j which C depends on. The disjunct element E_1 depends on the new branch point b_j, so the algorithm also makes this assertion. If adding E_1 to x causes a clash, E_1 is returned by the algorithm, as the class concept which has generated the clash.

As discussed previously in this section and in Section 3.4.2 current Tableaux reasoners employ dependency directed backjumping (Tsarkov et al., 2007, p. 294). Under this approach, when a clash detected by the Tableaux expansion tree (see Algorithm 3.1 earlier in this section), Tableaux backjumps to the earlier branch which the clash depends on and attempts to explore an alternative branch to alleviate the clash. This functionality is shown in Algorithmic form in Algorithm 3.3.

This algorithm, obtains the branch point node b_j, which the clashing concept $clash$ depends, given by $depOnBranch(clash)$. As discussed in Section 3.4.2 a branch has m number of possible expansions, each representing a separate ABox state. If any one expansion fails to generate a clash, then the model is considered consistent. The value m is equal to the number of disjunct elements contained in the disjunctive concept which gave rise to the branch point b_j, by application of the $\sqcup$-rule.

Let $HasNext(b_j)$ denote whether branch point b_j has any remaining unexpanded branches (disjunct elements). In the algorithm, if it is the case that

Algorithm 3.3 BackJump($clash$, $\mathcal{A}$, $\mathcal{G}$)

Inputs: ClassConcept $clash$, ABox $\mathcal{A}$, CompletionGraph $\mathcal{G}$
Outputs: Boolean $openBranchFound$

```
1: Let BranchNode b_j ← depOnBranch(clash)
2: while true do
3:    if b_j = null then
4:       return false
5:    end if
6:    *\ if b_j has unevaluated branches *\
7:    if HasNext(b_j) then
8:       RestoreTo(b_j, A, G)
9:       Let clashDetected ← ApplyNext(b_j)
10:      if clashDetected = false then
11:         return true
12:      end if
13:   else
14:      b_j ← depOnBranch(b_j)
15:   end if
16: end while
```

there are un-expanded branches, the reasoner state is restored its previous state when b_j was the active branch, using the $RestoreTo$ algorithm, which we will describe later in this section. Then the algorithm performs the next expansion from branch point b_j. Let $ApplyNext(b_j)$ perform this expansion, which is not shown. Assuming that b_j was created by applying $\sqcup$-rule to a disjunctive assertion $D(x)$, meaning that $createdByAssertion(b_j) = D(x)$, where D is a disjunction and x is an individual, then another unapplied disjunct element of D is added as a type for x. If this action results in a clash, then the concept which generated the clash is returned by $ApplyNext$. In the case that there are no more expansions for branch point b_j and $HasNext(b_j)$ returns $false$ then Algorithm 3.3 jumps back to an earlier branch point b_i where b_j depends on b_i. Then it attempts to expand any un-expanded branches for b_i, if there are any, otherwise it continues attempting to expand ancessorts of b_j, until the top most branch is reached (i.e. b_j is equal to $null$). If the top most branch is reached this means that there are no more branches to explore and that all branches led to a clash, thereby proving the inference being checked, holds.

The *RestoreTo* algorithm restores the reasoner state to an earlier state. The reasoner state is made up of the ABox and the $ToDo$ list of assertions which have transformation rules applicable to them. The ABox state is maintained using type and role labels $\mathcal{L}$. Therefore, the *RestoreTo* function restores all of the labels $\mathcal{L}$ and the $ToDo$ list to a previous state. Restoring to branch b_j means that if any assertion which was added to a type label $\mathcal{L}$ or $ToDo$ list at or after branch point b_j, is removed. In addition, if an assertion was removed from a label $\mathcal{L}$ or the $ToDo$ list at or after b_j, but was added before b_j, then this assertion is added back, by a restore. Such removals can be performed by transformation rules such as a merger of a pair of individuals x_r and x_p, by $\leq$-rule, which combines x_p into a individual x_r, and discards x_p. Any assertions which are removed during a restore are permanently discarded. This function is provided in Algorithm 3.4, which makes use of another sub-procedure provided in Algorithm 3.5.

Algorithm 3.4 RestoreTo(b_j, $\mathcal{A}$, $\mathcal{G}$)

Inputs: BranchNode b_j, ABox $\mathcal{A}$, CompletionGraph $\mathcal{G}$, where b_j is the branch node identifier to restore $\mathcal{A}$ to

1: **for all** $x_r \in \mathcal{A}$ where x_r is an individual **do**
2: $\quad\mathcal{L}(x_r) \leftarrow RestoreSet\big(\mathcal{L}(x_r), b_j\big)$
3: $\quad$**for all** $x_p \in \mathcal{A}$ where x_p is an individual **do**
4: $\quad\quad\mathcal{L}(\langle x_r, x_p\rangle) \leftarrow RestoreSet\big(\mathcal{L}(\langle x_r, x_p\rangle), b_j\big)$
5: $\quad$**end for**
6: **end for**
7: $ToDo \leftarrow RestoreSet(ToDo, b_j)$
8: **for all** $b_i \in \mathcal{G}$ **do**
9: $\quad$**if** $branchID(b_j) < branchID(b_i)$ **then**
10: $\quad\quad$remove b_i from $\mathcal{G}$
11: $\quad$**end if**
12: **end for**

Algorithm 3.4 obtains all of the labels and the $ToDo$ list passes these to Algorithm 3.5 which actually performs the restore to branch node b_j. Algorithm 3.4 does this by looping every individual in the ABox, passing the type label for each individual to *RestoreSet* as an input set S. In addition, for every individual, every other is looped, so that every role label can be obtained and

Algorithm 3.5 RestoreSet(S, b_j)

Inputs: Set S, BranchNode b_j
Outputs: Set S

1: **for all** $s_i \in S$ **do**
2: Let BranchNode b_s denote the $depOnBranch(s_i)$ for when element s_i was added to S
3: Let BranchNode b_r denote the $depOnBranch(s_i)$ for when element s_i was removed from S (or $branchID(b_r) = -1$, if not removed yet)
4: **if** $branchID(b_j) \leq branchID(b_s)$ **then**
5: remove s_i from S *s was added after b_j, remove it*\
6: **else if** $branchID(b_s) < branchID(b_j)$ and $branchID(b_j) \leq branchID(b_r)$ **then**
7: $S \leftarrow S \cup \{s_i\}$ *s_i was removed after b_j, add it back*\
8: **end if**
9: **end for**
10: **return** S

also passed to *RestoreSet* as an input set S. The $ToDo$ list is then passed to *RestoreSet* as an input set S.

Algorithm 3.5 removes any branch points in $\mathcal{G}$ which were added after b_j, removes any assertion which was added to the set S after b_j and adds back any assertion which was removed from S at or after b_j, but added before b_j, where b_j is the branch point node passed to *RestoreSet*.

In the next section we provide an example of the tree expansion for the standard Tableaux algorithm.

3.5.2 Tableaux Expansion Example

In this section provide an example which illustrates the operation of depth-first tree expansion and dependency directed backjumping. For this example, assume the ABox shown in Figure 3.14. In this ABox, let x_0, x_1 and x_2 be individuals, and let C_1, C_2 and C_3 be class concepts. The individual x_0 connects to x_1 and x_2 using the role R_1. The individual x_0 has the class type C_3, x_1 has the class type C_1 and x_2 has the class types C_1 and C_2.

We perform an inference check on this ABox to establish whether $C_0(x_0)$ holds, where C_0 is a conjunction of the form $C_0 \equiv (\exists R_1.(C_1 \sqcap C_2) \sqcap C_3)$. We

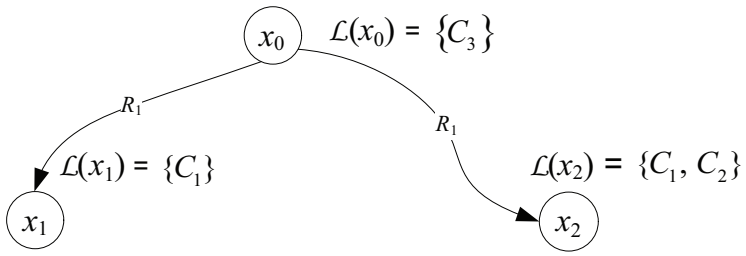


Figure 3.14: Tableaux Branching Example: ABox

illustrate the branch directed expansion which proves the inference in Figure 3.15. In the figure, the branch point identifier $branchID(b_k)$ for any given branch point b_k, is shown as subscript, where k is the identifier. In addition, shaded nodes represent nodes which were already created by a previous step in the Figure, while non-shaded nodes indicate those created by the current the step.

Since Tableaux proves inference by refutation, C_0 is transformed into a disjunction of the form $\neg C_0 \equiv (\forall R_1.(\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3)$. This disjunction is added at branch point b_1, therefore, $\neg C_0$ depends on branch b_1. In Figure 3.15, step a, the $\sqcup$-rule is applied to this disjunction, at branch point b_1. This results in the creation of a second branch point b_2 and the expansion of the first disjunct element $\forall R_1.(\neg C_1 \sqcup \neg C_2)$ of the disjunction, which is added as a type to x_0. The $\forall$-rule will then be applied to this disjunct element definition and will add $\neg C_1 \sqcup \neg C_2$ as a type to individuals x_1 and x_2 (this is not shown in the figure, because we are only focusing on tree expansion using the $\sqcup$-rule, the other transformations were explained in Section 3.4.2). The $\neg C_1 \sqcup \neg C_2$ definitions added to x_1 and x_2 depend on $\forall R_1.(\neg C_1 \sqcup \neg C_2)$ which depends on b_2. In step b, the $\sqcup$-rule is then applied to $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$. This creates a new branch node b_3 and adds the first element $\neg C_1$ as a type for x_1, which generates a clash. Since the clash was generated by $\neg C_1$, which depends on branch point b_2, the reasoner backjumps to b_2, meaning all assertions and branch point nodes after branch b_2 are removed. In step c, the next unapplied expansion for branch b_2 is then applied, which is second element $\neg C_2$ of the

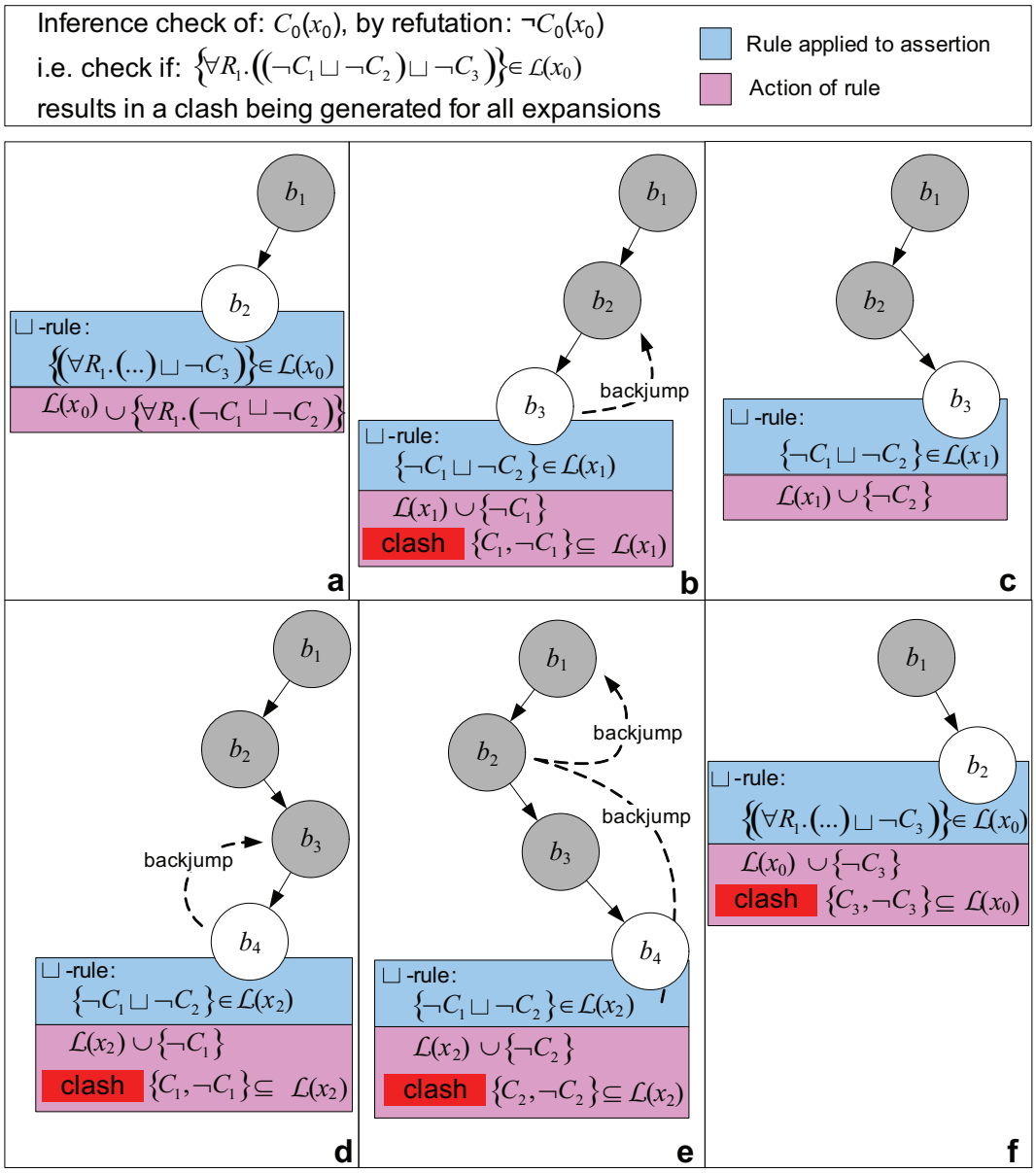


Figure 3.15: Tableaux Branching Example: Standard Tableaux Expansion Search Tree

disjunction $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$. This did not generate a clash. Therefore, in step d, the $\sqcup$-rule is applied to the next unapplied disjunction, which is $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$. This results in the creation of a new branch node b_4 and the first element $\neg C_1$ in the disjunction is add as a type for the individual x_2, which generates a clash. The concept $\neg C_1$ depends on b_3, so the reasoner backjumps to b_3. In step e the next unexpanded branch for b_3 is expanded, which is the second disjunct element $\neg C_2$ of the disjunction $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$.

Therefore $\neg C_2$ is added as a type for x_2, which generates a clash. There are no further expansions for the disjunction $\neg C_1 \sqcup \neg C_2$ at branch node b_3, and the disjunctino depends on branch b_2. Therefore, the reasoner backjumps to explore an alternative expansion of b_2. In step f, the expansion to branch node b_2 is evaluated, which is the second disjunct $\neg C_3$ from the disjunction $(\forall R_1.(\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3) \in \mathcal{L}(x_0)$. This results in adding $\neg C_3$ as a type to x_0 which generates a clash. There are no more unexplored branches, and all explored branches have now generated a clash, therefore, the inference is proven.

In this example we have illustrated Tableaux branch expansion for an inference check. It can be seen that expansion is depth-first, and the order in which a particular expansion possibility for a branch point node is expanded is arbitrary. However, when inference proof is used to match a user request against a service description, particular conditions in the request may have a different level of importance to the user. Therefore, in our proposed adaptive inference strategy, matching occurs in order of importance, rather than depth first / arbitrary. We will describe our adaptive strategy in Chapter 5. However, first, in the next chapter, we will describe our proposed approach to enable inference proof on-board a mobile device, using optimisation and caching strategies.

3.6 Summary

In the previous chapter we outlined the need to utilise semantic reasoning in order to provide accurate matching of user requests with service descriptions on-board small resource constrained mobile devices. However, semantic matching is considerably resource intensive. Therefore, in order to achieve efficiency we will propose optimisation and caching strategies in Chapter 4, as a key contribution of our thesis. In Chapter 5 we will also present our adaptive inference strategy which takes resources and user constraints into consideration

to enable priority based, incremental anytime matching. However, in order to understand these key contributions we first presented an overview of semantic inference provers used to complete semantic matching, in this chapter. Semantic inference provers reason with the OWL-DL language for the greatest accuracy while retaining computational completeness and this language is based on Description Logic (DL). Therefore, in this chapter, we also provided an overview of the DL notation. Finally, most DL inference provers utilise the Tableaux decision procedure to prove or disprove an inference. Therefore, our key contributions are based on the Tableaux algorithm. As a result, we also provided an overview of the Tableaux algorithm. In the next chapter we will outline our optimisation and caching strategies to enable on-board mobile inference.

Chapter 4

mTableaux: Light-Weight Mobile Inference

4.1 Introduction

In the previous chapter we provided an overview of the Description Logic (DL) language and the Tableaux algorithm, which is used in current reasoners to perform inference checks. As previously discussed Tableaux decision procedure is the most widely used DL reasoning algorithm (Baader et al., 2003, p. 322). However, semantic reasoning is considerably resource intensive and current architectures fail to provide efficient on-board mobile reasoning with expressive Description Logics (DL). Therefore, in this chapter we present our approach to enable light-weight mobile inference. We achieve this by proposing and developing optimisation and caching strategies which we call mobile Tableaux (mTableaux). The mTableaux algorithm and the research presented in this chapter have been published in Steller et al. (2009c); Steller and Krishnaswamy (2009, 2008a,c,b) and Steller et al. (2008).

The chapter is organised into the following sections. In Section 4.2 we provide an overview of our mTableaux optimisation strategies and in Sections 4.3, 4.4 and 4.5 we formally describe each of our mTableaux optimisation

and caching strategies, as the main contribution for this chapter. Finally, the chapter is summarised in Section 4.6.

4.2 mTableaux

In the following subsections we provide an overview of our mTableaux strategies by describing our optimisation strategies and our caching strategy.

4.2.1 mTableaux Optimisation Strategies

Our proposed mTableaux strategies enable on-board mobile service matching by optimising an inference proof which checks whether a user request matches a semantic service description. Tableaux is used to prove or disprove the inferred membership between a class concept definition and an individual. A class concept is a definition of one or more logical constraints which an individual must meet in order to be inferred as a type of this class concept. Therefore, in our approach a user request is defined using a class concept. An individual is used to represent a real world object and its properties. Therefore, in our approach a service definition is represented as an individual and its role relations. This follows the approach used to specify and match requests with services in (Stuckenschmidt and Kolb, 2008) using Description Logic (DL). As such, the focus of our approach is to provide efficient checking of the inferred relationship between a class concept and an individual. Moreover, Tableaux is used to check whether a request class concept C can be inferred to be a type for the service description individual x, written as $C(x)$, as described in Section 3.4.2. In order to improve the efficiency of the Tableaux decision procedure, we propose two heuristics to selectively control the application of Tableaux transformation rules:

1. **Selective application of transformation rules (ST)**: Rather than applying transformation rules to all assertions in $\mathcal{A}$, as in standard

Tableaux, these rules are applied only to membership assertions which relate to the service description individual x, or membership assertions about individuals which x connects to, using roles specified in the request definition C;

2. **Selective application of the disjunction transformation rule (SD)**: Rather than applying the $\sqcup$-rule to all disjunctive assertions in $\mathcal{A}$, as in standard Tableaux, the $\sqcup$-rule is applied only applied on those disjunctions which contain a reference to a class concept which also exists in the user request definition which is a class concept C;

For example, suppose the mobile user is searching for an Internet cafe, which is defined as any individual x which sells Internet and Coffee, where sells is a role, and Internet and Coffee are class concepts. The ST strategy, applies transformation rules to assertions about the individual x, and any individuals connected to x by the role sells. This is because the Internet and Coffee part of the request is relevant to individuals connected to x by role sells. ST does not apply transformation rules to any other individuals in $\mathcal{A}$ including those individuals which x connects to by roles other than sells such as hasPlayGround. Assume, the class concept Coffee is in fact a conjunction Coffee $\equiv$ InstantCoffee $\sqcap$ GroundCoffee. The SD strategy applies the $\sqcup$-rule to any assertion which contains Coffee, Internet, InstantCoffee or GroundCoffee, such as InstantCoffee $\sqcup$ GroundCoffee(x), but not to an assertion which does not, such as Tea $\sqcup$ HotChocolate.

Our optimisation strategies eliminate the application of transformation rules to some assertions. Since standard expansion of an ABox $\mathcal{A}$ involves exhaustively applying all transformation rules until no more rules can be applied, our ST and SD strategies do not fully expand the completion graph. As such, the ST and SD strategy do not guarantee completeness (Baader et al., 2003, p. 87) but retain soundness (Baader et al., 2003, p. 85). Moreover, when the ST and SD strategies are enabled they may fail to successfully prove that

a service description matches a user request which it does in fact semantically match (i.e. our strategies may produce false negatives). However, the strategies will never prove that a service description matches a user request which it does not semantically match (i.e our strategies will not produce false positives). This is because the Tableaux algorithm proves inferences by searching for contradictions. If the completion graph is not fully expanded, then these contradictions may not always be found. However, in the case that they are found, the positive inference is valid.

4.2.2 mTableaux Caching Strategy

Mobile users have certain interests, habits and preferences which typically influence their activities. As such, a user's previous activities are often a good indicator for which activities the user will carry out in the future (Kurkovsky et al., 2005). Therefore, a user may often perform requests for services which are similar to services requested in the past. Current reasoners treat every inference check as being independent of previous inference checks. Most OWL reasoners cache the result of inference checks rather than the evaluations which led to the match result and they generally do not store any cached information to secondary storage (Zacharias et al., 2007). However given the growth of secondary storage media / devices, this becomes a feasible option. Most small devices such as PDAs and mobile phones have secondary Secure Digital card memory which is inexpensive and can store several gigabytes of data. Therefore, we propose a caching strategy which records any matches found during previous requests to improve the response time of similar requests performed subsequently.

The Tableaux inference checking process is used in our approach to compare a service description against a user request. In some cases there is only one way of evaluating a request condition against a particular service description, and in other cases there are many ways of evaluating the request condition.

A particular request condition is fulfilled by the service description if every way of evaluating the condition matches the service description. Every time a match occurs, the definitions and assertions which have generated this match are stored in the cache.

During an inference checking process, the cache is checked to see if it contains the request condition currently being checked. There are two situations which may occur:

1. The request condition being compared against the service description is stored in the cache as having matched for every sub-condition of this condition. In this case, the condition is not evaluated again. Instead, it is assumed that the service description meets the request. However, since some request conditions are time sensitive, the user can specify that the cache may only be used in this way if the cache entry was stored after a specific time/day. For instance, the user may be searching for an Internet cafe which has a computer that is currently available for use. If the user sets an expiry time for a particular sub-condition then the time stamp for the cache entry must not exceed this limit in order to avoid re-evaluating it;

2. The request condition being compared against the service description is stored in the cache as having matched for some but not every sub-condition, or the user specified time limit for the condition was not met. In this case, the condition is re-evaluated. However, those definitions and assertions which previously contributed to the positive match finding, are evaluated in priority to other definitions and assertions. This allows the match to be found more efficiently for the current inference.

For instance, suppose a user requests a cafe which sells Tea and Coffee, where sells is a role, and Tea and Coffee are class concepts. Assume this is compared with a service description x which connects to an individuals y and

z using the role sells. Assume also that y has the type Tea and z has the types Tea and Coffee. During the inference checking process y is found to match the Tea condition and individual z matches both Tea and Coffee. Therefore, individual z matches for both possibilities / conditions in the request and individual y does not. In subsequent inference checks, individual z can be used to generate a clash immediately without needing to re-evaluate sells Tea and Coffee. Alternatively, if z did not exist, then y could be used, but since it did not generate a clash for all possibilities, it will be re-evaluated, but this evaluation will occur before evaluating over other individuals.

The order in which simultaneously applicable transformations are applied, does not impact on completeness or soundness (Tsarkov and Horrocks, 2005). Therefore, our caching strategy does not compromise completeness or soundness. However, if a match of a request condition to a service description is recorded in the cache, and the service description is subsequently changed, then this can lead to incorrect positive match findings (i.e. false positive). This is addressed by the ability for the user to specify a cache expiry time for request features / conditions. However, if the user does not specify an expiry for a cached request condition which has subsequently changed, then soundness is breached.

Our mTableaux strategies are focused on optimising a single inference check. Thus, the focus is on whether an individual x can be inferred to be a member of the class concept type C. The class C defines the user request and the individual x defines the a service description. As noted in Section 3.4.1, DL semantic reasoners perform classification (Baader et al., 2003, p. 72), (Tsarkov et al., 2007, p. 301) and realisation (Baader et al., 2003, p. 47). The classification phase compares every possible pair of class concept definitions from the TBox to see if they have an inferred subclass relationship. The realisation phase performs an inference check between every individual in the ABox and every class class in the TBox, to see if the individual is a member

of the class. Therefore, the realisation phase alone results in performing $m \times n$ inference checks where m is the number of classes in the ABox and n is the number of individuals in the ABox. This is despite the fact that the user is only interested in matching a single request concept with a single service individual. Performing inference checks for classes and individuals which are unrelated to the request or potential service does not add significant improvement / value to the service selection process. Therefore, our premise is that in order to improve performance efficiency in mobile environments these can be precluded. For instance, we have shown in our empirical evaluation in Section 6.4.2 that performing realisation significantly increases the response time required to perform an inference. Our architecture matches a request description C against several potential service descriptions, as separate successive inference operations.

In addition, current Tableaux reasoners perform a consistency check on an ontology before performing any inference checks about knowledge contained within the ontology. If it is found that an ontology is not consistent, then the reasoner reports an error and will not perform any inference checks on that ontology until the contradiction is resolved. This is because Tableaux performs inference checks by asserting the negation of an inference to the knowledge base and that proving a contradiction (clash) exists for every possible expansion. If this were performed on an inconsistent ontology, then false positive inference results would be found. As such, a consistency check only needs to be performed once, on a given set of knowledge contained in an ontology. In addition, a consistency check is resource intensive, because it involves applying all applicable transformation rules on any definition and assertion in the knowledge base, until no more transformation rules can be applied. Many of these transformation rules will be applied to definitions and assertions which do not relate to the user request or the service description being checked. Performing

ontology consistency checks has a high performance cost. In Figure 1.2 in section 1.2 we attempted to perform a consistency check for a realistic scenario on a mobile device and this failed to complete. As such, in the context of mobile semantic reasoning, performing an inference check on an inconsistent ontology to provide a result with reduced accuracy is still preferable to having no result at all. Therefore, since a consistency check is effectively a safety check, we do not perform this check on ontologies used by our approach. We make the assumption that ontologies containing service descriptions, have already been checked for consistency before they are made available for reasoning on a mobile device.

In the remaining sections of this chapter, we define each of our mTableaux strategies.

4.3 Selective Transformation Rule Application (ST)

As discussed previously, we use the Tableaux decision procedure to prove or disprove an inferred match. In order to ensure Tableaux inference proof is efficient in terms of time, in this section, we propose and develop our selective transformation rule application (ST) strategy. This strategy significantly optimises an inference check which is performed by the Tableaux decision procedure. As discussed in Section 3.4, Tableaux inference proof involves the application of transformation rules, which model the semantic interpretation of definitions and assertions. These rules are applied until the inference is proven or until no more transformation rules can be applied. Our proposed ST strategy, involves limiting the number of transformation rules applied to improve efficiency, while maintaining a high degree of accuracy. However, as stated earlier this strategy does not guarantee completeness, which means that not all positive inferences may be provable under this optimisation. However,

no negative inferences will be incorrectly proved as being a positive match through the use of this strategy. Since the ST strategy limits evaluation of some transformation rules it may not detect every possible clash, indicating a positive match for a particular request requirement. However, if a clash found then this is valid. Thus, the premise is that we reduce accuracy to improve performance efficiency, with the caveat that we aim to minimise the accuracy loss in the way we perform the optimisation.

4.3.1 Approach and Assumptions

In this section we describe the approach and assumptions of the Selective Transformation Rule Application (ST) optimisation strategy. In order to optimise the Tableaux inference proof procedure, this strategy eliminates the application of some transformation rules, to reduce the size of the proof task.

The goal of our proposed ST strategy, is to evaluate only a subset of the individuals in the ABox. The premise of our strategy is that during an inference check process which is comparing a request C_j with a service description x_i, we begin by evaluating only the service description x_i. When we encounter a universal quantifier definition of the form $\forall R.C$, where R is a role and C is a class concept, we need to further evaluate all R-neighbours of x_i. An R-neighbour of x is an individual y which x_i connects to by role R. We will now describe our approach in more detail.

As described previously, we are using the Description Logic (DL) language, in which knowledge is contained in a TBox $\mathcal{T}$ and ABox $\mathcal{A}$. The TBox $\mathcal{T}$ contains class concept definition, such that $\mathcal{T} = \{C_1, C_2, ..., C_m \mid (1 \leq j \leq m)\}$. The ABox $\mathcal{A}$ contains individuals x_i. These individuals can be asserted to be a type of a class concept C_j. In addition, a role can be asserted to connect one individual to another. These assertions are also contained in the ABox $\mathcal{A}$. As described in Section 3.4.2, ABox assertions are maintained using labels, such that class assertions $C_j(x_i)$ are expressed as $C_j \in \mathcal{L}(x_i)$ and role

assertions $R(x_i, x_k)$ are expressed as $R \in \mathcal{L}(\langle x_i, x_k \rangle)$. In standard Tableaux, a transformation rule is applicable to a class concept $C_j \in \mathcal{T}$ which has been asserted as a type for an individual $x_i \in \mathcal{A}$.

In our ST strategy, we evaluate only a subset of the individuals in the ABox $\mathcal{A}$. This means we apply transformation rules to class concepts asserted as types for only a subset of the individuals in $\mathcal{A}$. Let $\mathcal{ST}$ denote this subset of individuals in the ABox, such that $\mathcal{ST} \subseteq \mathcal{A}$. For instance, if x_1 is asserted to be a member of the classes C_1 and C_2, and $x_1 \in \mathcal{ST}$, then transformation rules can be applied to these two assertions $C_1(x_1)$ and $C_2(x_1)$. Otherwise, our strategy does not apply the rules. Note $\mathcal{ST}$ only contains individuals, not assertions.

Now the question remains as to how to decide which individuals x_i are to be included in the set $\mathcal{ST}$ for evaluation during the inference checking process. As described previously, Tableaux proves or disproves an inferred membership of an individual x_i to a class C_j, by adding the negation of class $\neg C_j$, to the type label for x_i. Transformation rules are then applied until it is established that a consistent $\mathcal{A}$ cannot be generated, which proves the inference. Therefore, $\mathcal{ST}$ must contain the individual x_i because it is clearly relevant to the inference check. Then during the inference checking process, neighbours of x_i are added to $\mathcal{ST}$ when a universal quantifier is encountered, where a neighbour is an individual which x_i connects to by some role R.

Adding additional individuals to $\mathcal{ST}$ is an iterative process as follows. A new iteration adds more individuals to $\mathcal{ST}$ when no more transformation rules can be applied, thus making more transformation rules applicable. In each iteration, if any individual $x_i \in \mathcal{ST}$ has a universal quantifier $\forall R_r.C$ asserted as a type in its type label, then any individual which x_i connects to using the role R_r, is also added to $\mathcal{ST}$. In addition, if any individual $x_i \in \mathcal{ST}$ has a universal quantifier $\forall R_p.C$ where R_p subsumes R_r, then any individual which x_i connects to by role R_p is also added to $\mathcal{ST}$. Adding a new iteration of

individuals to $\mathcal{ST}$ is specified in Equation 4.1, where x_i and x_y are individuals and R_r and R_p are roles.

$$\begin{aligned} \mathcal{ST} \;=\; & \mathcal{ST} \cup \bigcup_{x_i \in \mathcal{ST}} x_y, \text{ where} \\ & x_y \notin \mathcal{ST} \;\text{ and }\; \forall R_r.C \in \mathcal{L}(x_i) \;\text{ and} \\ & \Big(R_r \in \mathcal{L}(\langle x_i, x_y \rangle) \;\text{ or }\; R_p \in \mathcal{L}(\langle x_i, x_y \rangle) \;\text{ where }\; R_r \sqsubseteq R_p\Big) \end{aligned} \tag{4.1}$$

For example, a mobile user searching for an WiFi Internet cafe defines the request containing $\exists$ sells.(Internet $\sqcap$ WiFi) and this request is matched against the service description individual netcafe. This request definition is negated to give $\forall$ sells.($\neg$Internet $\sqcup$ $\neg$WiFi), and added as a type for the individual netcafe. Assume that netcafe is connected to another individual inet by the role sells, and inet is a member of the class types Internet and WiFi. Applying the $\forall$-rule on $\forall$ sells.($\neg$Internet $\sqcup$ $\neg$WiFi) will add the disjunction $\neg$Internet $\sqcup$ $\neg$WiFi to the type label of inet. inet will then need to be in $\mathcal{ST}$, in order to apply the $\sqcup$-rule to the disjunction which will generate the clashes required to prove the inference. This is why any individual to which netcafe connects by role sells, is added to the set of relevant individuals $\mathcal{ST}$.

We complete the process of applying all transformation rules before adding another iteration of individuals to $\mathcal{ST}$, because applying transformations to assertions about individuals which are closer to the service description individual x_i, are more likely to generate a clash, where x_i is the individual which is being checked for inferred membership to the request class C_j. Therefore, in each iteration, we add individuals which are $n+1$ number of role relations away from x_i, where n is the number of iterations performed. It is also noteworthy that, the application of transformation rules may add additional universal quantifiers $\forall R_p.C$ as types to individuals x_i already in $\mathcal{ST}$. If this occurs and then in that case that x_i is connected to any individuals x_y by role R_r, these x_y will be also added to $\mathcal{ST}$ at the next iteration.

The universal quantifier is the only definition type which is used for selecting new individuals to add to $\mathcal{ST}$. This is because of the effect of the $\forall$-rule which is applied to it. As described in Section 3.4.2, the only way an ABox can be changed is be the application of transformation rules. Since we are only interested in generating the clashes required to prove the inference, we only need to evaluate the individual being checked for inferred membership to the user request class, and any additional individuals which are changed by transformation rules acting on the user request. The $\forall$-rule applied to an individual x_i, asserts class types to individuals which x_i connects to by a role relation and these asserted class types may relate to the user request. Alternatively, the rules applied to a conjunction $C_1 \sqcap C_2$, disjunction $C_1 \sqcup C_2$, minimum cardinality restriction $\geq nR$, maximum cardinality restriction $\leq nR$ or existential quantifier $\exists R.A$ only make one of the following changes to the ABox:

- **Assert class concept definitions:** Class concepts are asserted as types to the individual they are applied to, which are already in $\mathcal{ST}$;
- **Create new individuals:** New individuals are created with empty type labels and no outgoing role relations, which implies that these new individuals have no asserted types to apply transformations to;
- **Combine existing individuals:** Generally, if individuals can be combined without generating an immediate clash, then applying transformation rules to these individuals will not cause a subsequent clash in typical mobile service matching scenarios;
- **Create a new individual with a single class type:** New individuals are created and a single class concept is added as a type to the individual. We assume that the concepts in the TBox do not contain any obvious contradictions such as $C \sqcap \neg C$. In the case that a transformation rule adds a consistent concept as a type to an individual with no other types or outgoing relations, it will not generate a clash.

In the following explain why each of the possible definition types (conjunction, disjunction, minimum / maximum cardinality restrictions, existential / universal quantifier) are used or not used for selecting new individuals to add to $\mathcal{ST}$:

- The $\forall$-rule is applied to a universal quantifier $\forall R.C$ which is asserted as a type to an individual x_i. It will add the additional class concept C to all neighbours which x_i connects to via role R. The concept C may be part of the user request and adding C to each neighbour may cause a clash to be generated. Therefore, all neighbours of x_i, which are connected by role R, are added to $\mathcal{ST}$ as discussed earlier in this section.

- The $\sqcap$-rule is applied to a conjunction $C_1 \sqcap C_2$ which is asserted as a type to an individual x_i. It will add more class concepts as types for x_i only. If the $\sqcap$-rule is being applied to a class concept added as a type to x_i, this means x_i is already in $\mathcal{ST}$. Therefore, no additional individuals should be added to $\mathcal{ST}$, due to the existence of a conjunction. For instance, assume a user wants to find a store serving either tea or coffee, which after negation becomes a conjunction of the form $\neg\mathsf{Tea} \sqcap \neg\mathsf{Coffee}$, compared against the service individual x_i. The $\sqcap$-rule will add both $\neg\mathsf{Tea}$ and $\neg\mathsf{Coffee}$ as a type label to x_i, but not any other individual.

- The $\sqcup$-rule is applied to a disjunction $C_1 \sqcup C_2$ which is asserted as a type to an individual x_i. It will add more class concepts as types for x_i only. If the $\sqcup$-rule is being applied to a class concept added as a type to x_i, this means x_i is already in $\mathcal{ST}$. Therefore, no additional individuals should be added to $\mathcal{ST}$, due to the existence of a disjunction. For instance, assume a user wants to find a store which sells Internet and coffee (an Internet cafe), which after negation becomes a disjunction of the form $\neg\mathsf{Internet} \sqcup \neg\mathsf{Coffee}$, compared against the service individual x_i. The

$\sqcup$-rule will add either $\neg\mathsf{Internet}$ or $\neg\mathsf{Coffee}$ as a type label to x_i, but not any other individual.

- The $\exists$-rule is applied to an existential quantifier $\exists R.C$ which is asserted as a type to an individual x_i. It will generate a new individual x_y and a connection from x_i to the new individual x_y using the role R. It will then add a class concept C to a new individual x_y. Since x_y is a new individual, it has no outgoing relations and C will be the only class type asserted to it. Therefore, if C is a consistent concept meaning it does not contain any obvious contradictions, then applying a transformation rule to it, will not generate a clash and x_y should not be added to $\mathcal{ST}$. As stated previously, we assume any downloaded ontology is consistent (i.e. it contains consistent concepts). We also assume that a valid request will not contain a condition as well as its negation such as $\neg\mathsf{Tea} \sqcap \mathsf{Tea}$, since this does not make sense. Therefore, all concepts in the TBox $\mathcal{T}$, including C, are assumed to be consistent. For instance, assume a request for a cafe contains $\forall\ \mathsf{sells}.(\mathsf{Coffee} \sqcup \mathsf{Tea})$, which when negated becomes an existential quantifier $\exists\ \mathsf{sells}.(\neg\ \mathsf{Coffee} \sqcap \neg\mathsf{Tea})$. If the request is being matched against an individual x_i, then this existential quantifier is asserted as an additional type to x_i. The $\exists$-rule will create a new individual x_y and will connect x_i to x_y using the role sells. It will then add $\neg\mathsf{Coffee} \sqcap \neg\mathsf{Tea}$ to the type label of x_y. Since x_y has no other types or role relations, there is nothing for $\neg\mathsf{Coffee} \sqcap \neg\mathsf{Tea}$ to clash with. The only way the $\exists$-rule will generate a clash is if it violates a $(\geq\ n\ R)$ definition which is also asserted as a type for x_i, however, x_i is already in $\mathcal{ST}$ meaning such a clash is already detectable by our strategy.

- The $\geq$-rule is applied to a minimum cardinality restriction $\geq\ n\ R$ which is asserted as a type to an individual x_i. It will create new individuals x_y and will connect x_i to the new individuals x_y using the role R, in order to meet the requirement that x_i must have least n number of R relations

to other individuals. Since the individuals are newly created, they will not have any class types or role relations asserted to them. Therefore, there are no transformation rules to apply to them. For instance, assume a user is searching for an appartment building which only has one floor, by specifying a maximum cardinality restriction, ≤ 1 hasFloor, which when negated becomes a minimum cardinality restriction ≥ 2 hasFloor. Assume, that x_i connects to an individual x_y using the role hasFloor. The $\geq$-rule will create an additional individual x_z to which x_i connects by role hasFloor. The only way this will generate a clash is if it violates a $\leq\ n\ R$ definition which is also added to the type label for x_i, however, x_i is already in $\mathcal{ST}$ meaning such a clash is already detectable by our strategy.

- The $\leq$-rule is applied to a maximum cardinality restriction $\leq\ n\ R$ which is asserted as a type to an individual x_i. It will merge pairs of individuals into a single individual, such that given the pair x_y and x_z, the individual x_z will be merged into x_y and x_z will be removed. This occurs to meet the requirement that x_i must have no more than n number of R relations to other individuals. In order for a merger of a pair to occur, the pair x_y and x_z cannot be declared as distinct from each other such that $x_y \neq x_z$. In the absence of any declaration between the two individuals, the merger can only take place if it does not generate a clash. In our proposed ST strategy, the merged individual x_y is not added to $\mathcal{ST}$. We suggest that when an individual connects to several individuals using the same role, these individuals are usually similar in their characteristics (defined in terms of which classes are added as types). Usually these individuals cannot be merged because they are explicitly declared as unequal, or their merger generates an immediate contradiction. For instance, if CarYard connects to BlueCar and RedCar, using the hasCar role, both cars are likely to be similar, unless there is a declaration such as BlueCar $\neq$ RedCar or

Blue $\neq$ Red, and Blue is added as a type to BlueCar and Red to RedCar. Either situation would generate an immediate clash. Therefore, we do not add successfully merged individuals to $\mathcal{ST}$, although this violates absolute completeness.

As described earlier in this section, a transformation rule can only be applied to a class membership assertion about an individual x_i iff $x_i \in \mathcal{ST}$. This condition is added as the first condition for each transformation rule as shown in Figure 4.1, where the other two conditions for each rule are the same as in standard Tableaux.

Furthermore, as described in Section 3.4.2 current Tableaux semantic reasoners employ the backjumping optimisation strategy. In this strategy, all assertions added to the type or edge label $\mathcal{L}$, and to the $ToDo$ list depend on a branch point node b_i in the expansion tree $\mathcal{G}$. When a backjump to branch point b_i occurs, all assertions which were added at or after b_i, are removed. That means, a branch point b_{i+1} with a level equal to or exceeding b_i, is removed. As a result, our $\mathcal{ST}$ set of individuals which can be evaluated, must also support the backjumping functionality. Therefore, when an individual x is added to $\mathcal{ST}$ it depends on the most recent branch point node b_i which has the highest level in $\mathcal{G}$. In the case that a backjump to branch point b_i occurs, any individual $x \in \mathcal{ST}$, which depends on a branch point b_{i+1} (where b_{i+1} has a level which is equal to or higher than b_i) is removed from $\mathcal{ST}$.

In the next section we will illustrate the operation of the ST optimisation strategy, using the example from Section 3.4.3.

4.3.2 Example

In this section we refer to the inference checking example which was provided in section in Section 3.4.3. This example checks whether the class concept C_0 definition can be inferred to be a type for the individual x_0. In our mTableaux approach, the inference checking process is used to check whether the user

$\sqcap$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $C_1 \sqcap ... \sqcap C_n \in \mathcal{L}(x_i)$, and
3. $\{C_1, ..., C_n\} \nsubseteq \mathcal{L}(x_i)$

then $\mathcal{L}(x_i) \leftarrow \mathcal{L}(x_i) \cup \{C_1, ..., C_n\}$

$\sqcup$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $C_1 \sqcup ... \sqcup C_n \in \mathcal{L}(x_i)$, and
3. $\{C_1, ..., C_n\} \cap \mathcal{L}(x_i) = \emptyset$

then $\mathcal{L}(x_i) \leftarrow \mathcal{L}(x_i) \cup \{C\}$ for some $C \in \{C_1, ..., C_n\}$

$\exists$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $\exists R.C \in \mathcal{L}(x_i)$, and
3. there is no R-neighbour x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $C \in \mathcal{L}(x_j)$

then create a new node x_j with $\mathcal{L}(\langle x_i, x_j \rangle) = \{R\}$ and $\mathcal{L}(x_j) = \{C\}$

$\forall$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $\forall R.C \in \mathcal{L}(x_i)$, and
3. there is some R-neighbour x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $C \notin \mathcal{L}(x_j)$

then $\mathcal{L}(x_j) \leftarrow \mathcal{L}(x_j) \cup \{C\}$

$\geq$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $\geq\ n\ R \in \mathcal{L}(x_i)$, and
3. there are no R-neighbours x_j of x_i such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$ where $x_j \neq x_k$, $1 \leq j \leq k \leq n$

then create new nodes x_j with $\mathcal{L}(\langle x_i, x_j \rangle) = \{R\}$, and $x_j \neq x_k$, $1 \leq j \leq k \leq m$, so that $m = n$

$\leq$-rule: if
1. $x_i \in \mathcal{ST}$, and
2. $\leq\ n\ R \in \mathcal{L}(x_i)$, and
3. there are R-neighbours x_j of x_i, such that $R \in \mathcal{L}(\langle x_i, x_j \rangle)$, $1 \leq j \leq m$, where $m > n$ and there is a pair of R-neighbours which is not declared $x_j \neq x_k$, $1 \leq j \leq k \leq m$

then for some pair x_j, x_k which is not declared $x_j \neq x_k$, $Merge(x_j, x_k)$ for $1 \leq j \leq k \leq m$
/*for $Merge$ see Figure 3.10, Section 3.4.2*/

Figure 4.1: mTableaux Transformation Rules for the ST Optimisation Strategy

request C_0 matches the service description x_0. In this inference check, C_0 is negated and added as a type to x_0. If all attempts to expand the ABox using transformation rules, fail due to a contradiction, then the inference is proven. The negation of the definition C_0, added as a type to x_0 is $\neg C_0 \equiv \forall R_2.(\leq 0R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$. Figure 4.2 shows the example from Section 3.4.3, however, we highlight in yellow, the transformations which are performed when using our proposed ST strategy performs. Any transformations which are eliminated by our ST strategy, have a red line through them. We also show new iterations of individuals being added to the set $\mathcal{ST}$. Only those individuals which are contained in $\mathcal{ST}$, are evaluated by our strategy. Initially, $\mathcal{ST}$ contains only the individual x_0, because this is the service description individual which the inference check is about. Thus, it is clear that in our approach reduces the size of the matching problem, which we hypothesise will significantly improve response time.

As shown in the figure, all definitions which are added as types to x_0 are evaluated first. The $\forall$-rule is not applied to $\forall R_3.(\neg C_1 \sqcup \neg C_2)$ which is added to x_6, because x_6 is never added to the set $\mathcal{ST}$. The $\sqcup$-rule is applied to $\forall R_2.(\leq 0R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3)) \in \mathcal{L}(x_0)$, and the $\forall$-rule to $\forall R_2.(\leq 0R_2) \in \mathcal{L}(x_0)$, as in the standard Tableaux example in Section 3.4.3. At this point a clash has not been generated, and there are no more rules to apply on the individuals in $\mathcal{ST}$, which includes only x_0 at this stage. Therefore, another iteration of individuals are added to $\mathcal{ST}$. Since x_0 has the universal quantifier definition $\forall R_2.(\leq 0R_2)$ as a type, the individual x_1 which x_0 connects to by role R_2 is added to $\mathcal{ST}$ after no more transformation rules can be applied to the individuals already in $\mathcal{ST}$. Now the $\leq$-rule can be applied to $\leq 0R_2$ which is added to x_1, and this generates a clash. This process continues until all clashes are detected and the inference is proven.

As is shown in Figure 4.2, the individuals which were not evaluated include x_2, x_6 and x_7. That is, transformation rules were not applied to definitions

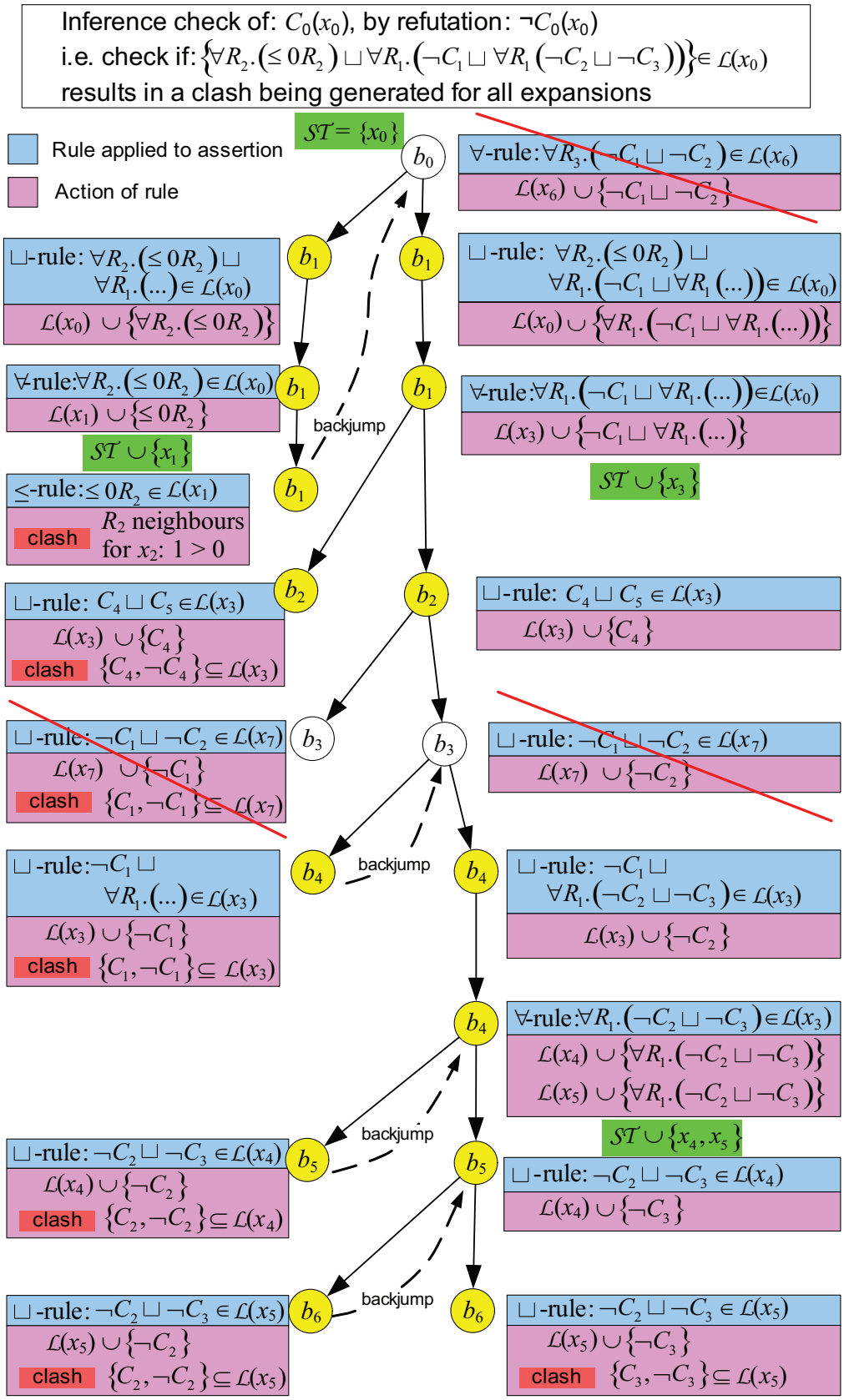


Figure 4.2: Selective Transformation (ST) Optimisation Strategy Example

added as types to those individuals. This is because none of the individuals contained in the set $\mathcal{ST}$ connected to x_2, x_6 or x_7 using a role that was specified in a universal quantifier, added to them as a type. For instance, the disjunction $\neg C_1 \sqcup \neg C_2$ added to the individual x_7 was not applied. If applied, this disjunction would only have generated a clash for only the first expansion and not the second, and therefore, would not have contributed to the inference proof. In larger examples, we hypothesise that eliminating transformations which are unlikely to affect the outcome of an inference check will lead to a significant performance gain, with minimal impact on accuracy.

In this section we presented our ST optimisation strategy, to enable more efficient matching of service descriptions x with user requests C, using a Tableaux inference check. Our ST strategy eliminates the application of some Tableaux transformation rules to assertions based on which individuals these assertions are about. That is, transformation rules are only applied to a subset of individuals in the ABox, where $\mathcal{ST}$ is a set containing this subset. However, an individual $x_i \in \mathcal{ST}$ may still have many disjunctions added to its type label. As discussed in Section 3.4.2, the $\sqcup$-rule is non-deterministic, meaning that it generates a number of possible / alternative ABox expansions, thereby increasing the size of the search space (Tsarkov and Horrocks, 2005). Therefore, reducing the number $\sqcup$-rule applications considerably reduces the size of the inference task.

Some of the disjunctions added to the type label of an individual $x_i \in \mathcal{ST}$ may not be relevant to the inference check comparing the user request with a service description. Therefore, in the next section we propose our Selective Disjunction Rule Application (SD) optimisation strategy, which complements our ST strategy. We hypothesise that our SD strategy will add to the improvements in efficiency by further reducing the number of transformation rule applications.

4.4 Selective Disjunction Rule Application - (SD)

In this section we propose and develop our Selective Disjunction Rule Application (SD) strategy. The SD strategy eliminates the application of the ⊔-rule to some disjunctions to improve efficiency, while maintaining a high degree of accuracy. This strategy, does not guarantee completeness, which means that not all positive inferences may be provable. However, as stated in Section 4.2, no negative inferences will be incorrectly proven as being a positive match through the use of this strategy. This is because Tableaux proves inferences by searching for clashes. Since the SD strategy limits the evaluation of some ⊔-rule applications, it may not detect all possible clashes, which prove a positive match for a particular feature in the user request. However, if a clash found by our strategy, then this is valid. Thus, we reduce accuracy in order to improve performance efficiency, however, we aim to minimise any loss to accuracy.

4.4.1 Approach and Assumptions

Our SD optimisation strategy proposes an approach to improve the efficiency of a Tableaux inference check, without significantly reducing accuracy. This strategy achieves increased efficiency by eliminating the evaluation of some disjunctions. During the inference checking process, when a disjunction is encountered, it is only evaluated if it contains a class concept that is also found as part of the user request.

The reason our proposed SD strategy seeks to reduce the number of disjunctions which are evaluated, is due to the fact that evaluating a disjunction, by application of the ⊔-rule, significantly increases the size of the search

space (Tsarkov and Horrocks, 2005). This occurs because the $\sqcup$-rule is non-deterministic, and generates a finite number of expansions which must each be explored to prove that a non-contradictory expansion cannot be constructed.

Our proposed SD strategy evaluates only those disjunctions which contain a class concept which is also found somewhere in the user request. This strategy based on the premise that an ABox is assumed to be consistent, before the negation of the conjunctured inference is asserted to the ABox. That is, as previously discussed, Tableaux proves that a potential service individual x matches a user request class definition C by asserting $\neg C$ as a type for x. The inference is proven when every attempt to fully expand $\mathcal{A}$ into a consistent $\mathcal{A}'$ fails, due to the occurrence of a contradiction. Therefore, the subsequent failure to construct a consistent expanded ABox $\mathcal{A}'$ is directly attributable to the inclusion of the negation $\neg C$. For instance, assume a mobile user is requesting an Internet cafe defined as $\exists$sells.Internet and $\exists$sells.Coffee. During the inference process, a disjunction of the form Tea $\sqcup$ Coffee can be evaluated, because Coffee also appears in the user request. However, a disjunction Pizza $\sqcup$ VideoGame, need not be be applied because neither Pizza nor VideoGame appear in the request.

The decision about whether or not to evaluate disjunctions is thus a two stage process:

1. At the beginning of the inference check process, all the class concept definitions specified in the user request are extracted and stored in a set of class concepts, let $\mathcal{SD}$ denote this set;

2. During the inference checking process, when a disjunction is encountered, all the class concept definitions specified in the disjunction are extracted. If any of the extracted definitions are contained in $\mathcal{SD}$, then the $\sqcup$-rule is applied to the disjunction, otherwise it is not.

First we will describe the process of extracting the class concepts from the user request class concept. The class definitions are extracted from the request

definition and added to the set $\mathcal{SD}$ using Algorithm 4.1. The set $\mathcal{SD}$ is filled using the call $GenSD(\neg C)$ where C is the user request. We negate the request C because its negation will be added as a type for the service description x.

Algorithm 4.1 GenSD(C_i)

Inputs: ClassConcept C_i
Outputs: Set $\mathcal{SD}$

Let $\mathcal{SD} \leftarrow \emptyset$
if $C_i = \neg C_j$ **then**
  $\mathcal{SD} \leftarrow \{C_j\}$ *remove any leading negations*\
else
  $\mathcal{SD} \leftarrow \{C_i\}$
end if
if $C_i = \forall R.C_j$ **then**
  $\mathcal{SD} \leftarrow \mathcal{SD} \cup GenSD(C_j)$
else if $C_i = \exists R.C_j$ **then**
  $\mathcal{SD} \leftarrow \mathcal{SD} \cup GenSD(C_j)$
else if $C_i = C_1 \sqcap ... \sqcap C_m$ **then**
  for all C_j from $C_1 \sqcap ... \sqcap C_m, 1 \leq j \leq m$ **do**
    $\mathcal{SD} \leftarrow \mathcal{SD} \cup GenSD(C_j)$
  end for
else if $C_i = C_1 \sqcup ... \sqcup C_m$ **then**
  for all C_j from $C_1 \sqcup ... \sqcup C_m, 1 \leq j \leq m$ **do**
    $\mathcal{SD} \leftarrow \mathcal{SD} \cup GenSD(C_j)$
  end for
end if
Let W contain all subclasses of C_j, such that $C_k \sqsubseteq C_j$ for all $C_k \in W$
$\mathcal{SD} \leftarrow \mathcal{SD} \cup W$
return $\mathcal{SD}$

The algorithm receives a class concept C_i as an input parameter, where C_i is initially the user request. C_i is added to the set $\mathcal{SD}$, after removing any leading negations. We remove negations from all class concept definitions in $\mathcal{SD}$ (and from all concepts extracted from disjunctions) because, in this proposed optimisation, a concept C compared to a concept $\neg C$ should be considered equal. For instance, if a disjunction contains $\neg C$, and $C \in \mathcal{SD}$, the disjunction is still applied. If the class concept C_i is actually a universal quantifier $\forall R.C_j$ or existential quantifier $\exists R.C_j$ definition then C_j is extracted and passed back to $GenSD$ for evaluation using a recursive call, and any class concepts which have been extracted, and returned by $GenSD$ are added to

$\mathcal{SD}$. If the class concept C_i is a conjunction or disjunction, each conjunctive or disjunctive class concept element is passed back to $GenSD$ using a recursive call and the extracted concepts are added to $\mathcal{SD}$. The reason we make recursive callbacks to $GenSD$, is because the class concept definitions C_j passed back to $GenSD$, all relate to the user request, and these class concepts C_j may be asserted as a type to x or another individual by a transformation rule as described in Section 3.4.2. In addition, all subclasses of the input concept C_i are also added to $\mathcal{SD}$. For instance, if the user request for an Internet cafe is defined as $\exists$ sells.(Coffee $\sqcap$ Tea), this is negated to become $\forall$ sells.($\neg$Coffee $\sqcup$ $\neg$Tea), and added to the service description x. Applying the $\forall$-rule to this definition, will add $\neg$Coffee $\sqcup$ $\neg$Tea to any individual y, which connects from x by the role sells. Thus, $GenSD$ must extract Coffee and Tea and add these to $\mathcal{SD}$, so that the disjunction $\neg$Coffee $\sqcup$ $\neg$Tea which is relevant to the user request, can be evaluated by this proposed SD strategy.

Now that we have described how the set $\mathcal{SD}$ is populated, we describe how the decision to evaluate a disjunction is made using this set. Our proposed SD strategy deems a disjunction D applicable for evaluation if any of its disjunctive elements are contained in $\mathcal{SD}$, as defined in Algorithm 4.2. This algorithm is initiated by a the call $ApplyDisj(D, \mathcal{SD})$ where D is the disjunction to check.

Algorithm 4.2 ApplyDisj($D, \mathcal{SD}$)

Inputs: ClassConcept D, Set $\mathcal{ST}$
Outputs: Boolean $applyDisj$

```
1: for all C_i from D where D = C_1 ⊓ ... ⊓ C_n, 1 ≤ i ≤ n do
2:   if C_i ∈ SD then
3:     return true
4:   else
5:     Let W contain all subclasses of C_i, such that C_j ⊑ C_i for all C_j ∈ W
6:     if C_j ∈ ST for any C_j ∈ W then
7:       return true
8:     end if
9:   end if
10: end for
11: return false
```

The algorithm loops through each class concept element of the disjunction and if any of these elements, or a subclass of these elements are contained in $\mathcal{SD}$ then the disjunction is evaluated. In other words, the $\sqcup$-rule can be applied to such a disjunction. Figure 4.3 illustrates the modified $\sqcup$-rule definition, which contains an extra condition specifying that $ApplyDisj(C_1 \sqcup ... \sqcup C_n, \mathcal{SD})$ must return *true*, in order to apply the disjunction $C_1 \sqcup ... \sqcup C_n$. The other two conditions are the standard Tableaux conditions for the $\sqcup$-rule as stated in Section 3.4.2. It is noteworthy that if our proposed SD strategy is used in conjunction with our ST strategy (see Section 4.3), then the $\sqcup$-rule will also include a fourth condition, which was specified in Section 4.3, requiring that the individual x_i must be contained in the set $\mathcal{ST}$, where $C_1 \sqcup ... \sqcup C_n$ was added to the type label of x_i.

$\sqcup$-rule: if
1. $ApplyDisj\big((C_1 \sqcup ... \sqcup C_n), \mathcal{SD}\big) = true$, and
2. $C_1 \sqcup ... \sqcup C_n \in \mathcal{L}(x_i)$, and
3. $\{C_1, ..., C_n\} \cap \mathcal{L}(x_i) = \emptyset$

then $\mathcal{L}(x_i) \leftarrow \mathcal{L}(x_i) \cup \{C\}$ for some $C \in \{C_1, ..., C_n\}$

Figure 4.3: mTableaux $\sqcup$-rule Transformation for SD Optimisation Strategy

In the next section we will illustrate the operation of the SD optimisation strategy using the example from Section 3.4.3.

4.4.2 Example

In this section we refer to the inference checking example which was provided in section in Section 3.4.3. This example checks whether the class concept C_0 definition can be inferred to be a type for the individual x_0. In our architecture, the inference checking process is used to check whether the user request C_0 matches the service description x_0. In this inference check, C_0 is negated and added as a type to x_0. If all attempts to expand the ABox using transformation rules fails due to a contradiction, then the inference is proven. The negation

of the definition C_0, added as a type to x_0 is $\neg C_0 \equiv \forall R_2.(\leq 0R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$.

When our proposed SD strategy is enabled, it must first extract all the class concept definitions from the user request and add these to the set $\mathcal{SD}$. The contents of $\mathcal{SD}$ extracted from $\neg C_0$, using the algorithm *GenSD* (see Algorithm 4.1 defined in the previous section), is shown in Figure 4.4.

$\mathcal{SD}$
$\forall R_2.(\leq 0R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$
$\forall R_2.(\leq 0R_2)$
$\leq 0R_2$
$\forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$
$(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$
C_1
$\forall R_1.(\neg C_2 \sqcup \neg C_3)$
$(\neg C_2 \sqcup \neg C_3)$
C_2
C_3

Figure 4.4: Example Contents of the Set $\mathcal{SD}$

During the inference checking process a disjunction is only evaluated if one of its disjunctive members are contained in $\mathcal{SD}$. Figure 4.5 illustrates the ABox expansions involved to check the validity of the inferences by applying transformation rules. This figure is identical to Figure 3.12 from Section 3.4.3, which illustrates the inference checking process using standard Tableaux, except that only the transformations which are highlighted in yellow are applied. Any disjunctions which are eliminated by our SD optimisation strategy, are shown in the figure as being crossed out with a red line. As is shown in the figure, the $\sqcup$-rule is not applied to the disjunction $C_4 \sqcup C_5$, because neither C_4 nor C_5 are contained in the set $\mathcal{SD}$. If applied, this disjunction would have generated a clash for C_4, but not C_5, and therefore, would not contribute to the inference proof. The example still successfully proves that C_0 can be inferred to be a type for x_0. Therefore, in this example, our SD optimisation has reduced the number of evaluations required to prove the inference to improve response time without reducing accuracy. In realistic scenarios there will be

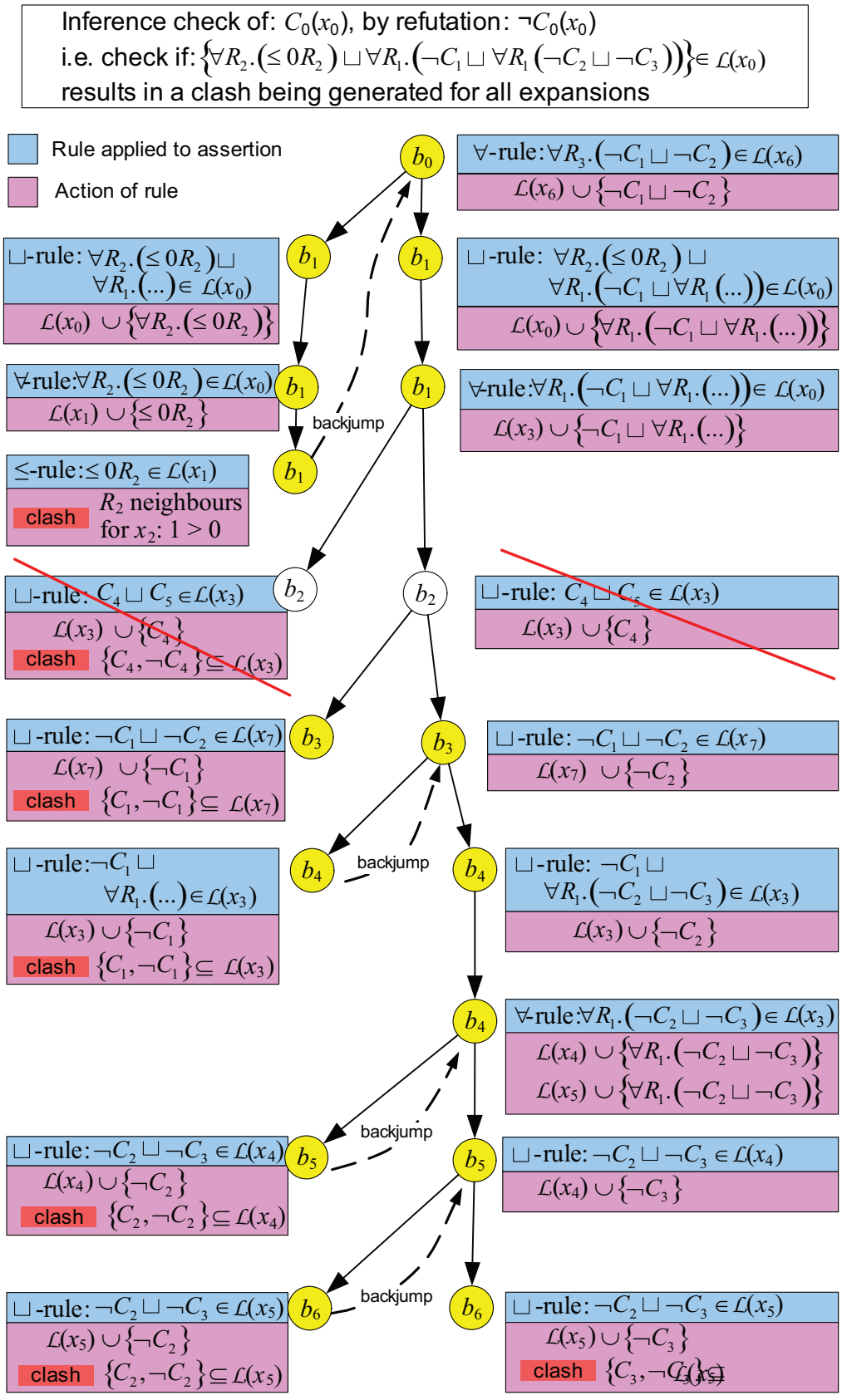


Figure 4.5: Selective Disjunction (SD) Optimisation Strategy Example

many individuals which have disjunctive definitions asserted to them as types. Therefore, we hypothesise that this optimisation will improve response time while maintaining a high / acceptable level of accuracy for reliable semantic reasoning.

When the ST and SD optimisation strategies are used together, they eliminate the application of transformation rules to certain definitions. However, even when both optimisation strategies are enabled, there are some transformation rules applied which do not assist in proving the inference. For instance, in the example the $\forall$-rule was applied to $\forall R_1.(\neg C_2 \sqcup \neg C_3)$, resulting in $\neg C_2 \sqcup \neg C_3$ to be asserted as a type for individuals x_4 and x_5. This was because x_3 connects to both x_4 and x_5, using role R_1. Since both disjunctions depend on $\forall R_1.(\neg C_2 \sqcup \neg C_3)$, only one disjunctions needs to generate a clash for both expansions in order to prove that this universal quantifier generated a clash. However, there is no way of knowing which disjunction will generate a clash for all expansions. In realistic data sets, x_3 may connect to many more individuals using role R_1 than just x_4 and x_5. Determining which disjunction generates a clash for all expansions, may thus result in significant processing. However, previous requests may help to indicate which of these disjunctions should be evaluated first. Thus, a cache of all assertions which have generated a clash during previous inference checks can help to improve response time. In addition, if particular parts of a service description are unlikely to change frequently, such a cache could be used to avoid some expansions all together. In the next section we will propose our caching strategy, which addresses this need.

4.5 Caching Strategy (CS)

In the previous sections we have described our proposed strategies for optimising a Tableaux inference check which is used by our mTableaux to compare a service description with a mobile user's request. In this section we outline

a strategy which caches previous user requests and uses this information to reduce the response time of subsequent requests.

Entries in the cache are used to avoid re-evaluating parts of an inference check which have already been evaluated in previous inference checks. Therefore, our proposed CS does not compromise completeness or soundness, unless the ontology has changed since the entry has been stored in the case. However, to alleviate the limitation of out of date information, we allow users to associate a time limit with request attributes. If an entry relating to a condition was stored in the cache before this user specified time, then the condition must be re-evaluated, to ensure completeness and soundness for this condition. In the next section we will detail our proposed strategy. In the case that the user does not specify expiry times for the conditions in the request, our proposed CS strategy may prove inferences which are no longer positive if the description of a service has changed since it was cached. However, this strategy will not prove negative inferences because only positive match results are stored in the cache. If a request is not found in the cache then the standard Tableaux matching process occurs. In the next section we will detail our propose strategy.

4.5.1 Approach and Assumptions

The daily lives of users are influenced by their interests, habits and preferences. It is, therefore, likely that users will request services which are similar to services requested in the past. In this section, we propose a caching strategy which records matches found during previous requests to improve the response time of similar requests performed subsequently. For instance, a user may have previously made a request to find a store in a shopping centre which sells CDs relating to particular style of music. The user may subsequently seek similar CDs from the same genre of music in the future (Kurkovsky et al., 2005).

Caching mechanisms have been used by some reasoners to cache the results of particular inference checks in main memory (Tsarkov et al., 2007). However, current reasoners treat every inference check as being independent of previous inference checks. Most OWL reasoners cache the result of inference checks in memory only and do not store these to secondary storage (Zacharias et al., 2007). However, today's mobile devices such as PDAs and phones have substantial amounts of secondary Secure Digital card memory which is inexpensive and can store several gigabytes of data. Therefore, we hypothesise that a caching strategy that leverages secondary storage will improve response time in a mobile setting, where processing power and main memory is resource constrained, by avoiding re-processing of requests which are similar to those performed in the past. Current reasoners do not cache the transformation rule applications which generated clashes. The novel aspect of our caching strategy (CS) is that we persistently cache the Tableaux transformations which have generated a clash during an inference check. Depending on the level of completeness required for a particular condition in the user request, our strategy uses a cached entry to:

1. generate an immediate clash during an inference check;

2. re-order the application of transformation rules, such that those which are stored in the cached entry are applied first.

We will describe our caching strategy (CS) in more detail in the remainder of this section.

In our proposed CS strategy, when a part of the user request is found to match a particular service description, the definitions and assertions which were evaluated to come to this finding, are stored in a cache. Whenever a definition or assertion is evaluated in subsequent inference checks, our CS strategy will check to see if these are contained in the cache. If this is found to be the case, then either these definitions and assertions will not need to

be re-evaluated, or they will be evaluated before others, in order to improve response time (depending on the time validity of the cached entry).

While most definitions and assertions which define a service, are generally static or do not change very often, it is possible that some parts of the service description have changed since the last time a request was compared against this service. In other words, it is possible some entries in the cache can become out of date. Therefore, we associate a time stamp with every entry in the cache. The user can also associate a time stamp with particular time sensitive request conditions. If the cache entry was stored before the user specified time limit (i.e. expiry time) for a particular request condition, then this condition must be re-checked against the service description, to ensure accuracy. However, the definitions and assertions which were evaluated to find the match previously, will be given increased priority, and evaluated before other definitions and assertions. For instance, a user may request an Internet cafe, which has a computer that is currently available for user. The part of the request which defines an Internet cafe is considered unlikely to change, while the availability condition is considered time sensitive and must be re-evaluated if last checked, over 15 minutes ago.

In addition, we discussed in Section 3.4.2, there may be more than one way of evaluating a request condition. For instance, a negated request for an Internet cafe may contain the definition $\forall$sells.Coffee, where sells is a role and Coffee is a class concept. The Coffee class concept may be defined in the TBox $\mathcal{T}$ as InstantCoffee $\sqcap$ GroundCoffee. This implies that the coffee requirement is only satisfied if both InstantCoffee and GroundCoffee are met. However, in our proposed caching strategy (CS), every match is stored in the cache. Therefore, if the InstantCoffee was found to match a service description, we store this in the cache, even if GroundCoffee did not match the service. This is because the user may subsequently perform a request which contains InstantCoffee but does not require GroundCoffee. We will also store the Coffee class concept in the

cache, but only as a partial match. Thus, in our cache we distinguish between a full and partial match. A particular request condition does not need to be re-evaluated if it is contained in the cache as a full match (and has not expired according to the time validity).

Since our proposed CS strategy supports time bounded request conditions and differentiates between a full and partial match in the cache, this means that one of two situations may occur:

1. In the case that a request condition is stored in the cache as a full match and falls within the user specified time limit, then the condition is assumed to match and is not re-evaluated;

2. In the case that a request condition is stored in the cache as a partial match or its time stamp exceeds the user specified time limit, then the condition must be re-evaluated. However, all definitions and assertions which gave rise to the (partial) match finding previously, are stored in the cache and given priority so that they are applied before others.

Now that we have provided an overview of our proposed CS strategy, we will provide a more formal description of the structure of the cache in the next section.

4.5.2 Cache Structure

As described in Section 3.4.2, Tableaux proves an inferred match between a user request and service description, by asserting the negation of the request to the service individual, and applying transformation rules which generate expansions, until a contradiction is established for every possible expansion. Thus, whenever a clash occurs, this contributes to a match for a condition in the request. As such, whenever a clash occurs, we store the definitions and assertions which have generated the clash in the cache. This means storing the definition which contradicts another definition, plus all the definitions which

the clashing definition depends on. In Tableaux, a class concept definition C_j asserted as a type for an individual x, may depend on another concept or a branch point node b in the expansion tree $\mathcal{G}$. A branch point node b is created by a non-deterministic transformation rule such as the $\sqcup$-rule which is applied to a disjunction $C_i \sqcup C_j$ which is a type for the individual x. When a transformation rule is applied to a class concept C_j which generates an obvious contradiction, then C_j is stored in the cache as a full clash (i.e. a full match). In addition, C_j may depend on another class concept C_k. Alternatively, C_j may depend on a branch node b, where b was created by applying a disjunction $C_k \sqcup C_p$ which is a type for the individual x and the disjunct element C_k was the element which was last evaluated (i.e. added as a type to the individual x). In either case, we say that the assertion $C_j(x)$ depends on $C_k(x)$. All assertions which $C_j(x)$ depends on are also stored in the cache. These dependencies are marked in the cache as partial clashes (i.e. a partial match), unless all possible expansions have been evaluated and have generated a clash.

The cache for our proposed CS strategy, is structured as a set of tuples, which take the form shown in Equation 4.2. In this equation: $C_i(x_r)$ and $C_j(x_p)$ are assertions. The second element $C_j(x_p)$ is an assertion which depends on the assertion in the first assertion $C_i(x_r)$. t is a time stamp indicating when the cache entry was added. s indicates the status of the clash which was generated, which is either $true$ for a full clash (i.e. a full match for $C_i(x_r)$) or $false$ for a partial clash (i.e. a partial match for $C_i(x_r)$).

$$
\begin{aligned}
&CSCache = \{\zeta_1, ..., \zeta_n\} \quad \text{where} \\
&\quad \zeta = \left\langle C_i(x_r), C_j(x_p), t, s \right\rangle \quad \text{such that} \\
&\quad C_j(x_p) \text{ is assertion which depends on the assertion } C_i(x_r), \\
&\quad t \text{ is the date/time when the tuple } \zeta \text{ was added to } CSCache \\
&\quad s \text{ is } true \text{ if all possibilities for } C_i(x_r) \text{ clashed or } false \text{ otherwise}
\end{aligned}
\tag{4.2}
$$

For example, suppose there a negated request for an Internet cafe ¬ UserRequest which contains ¬Coffee ⊔ ¬InternetRequest. In addition, assume ¬InternetRequest is a disjunction, such that ¬InternetRequest ≡ (∀ hasComm.¬WiFi ⊔ ¬Internet). Suppose that the class concept ¬WiFi generated a clash. In this case, ¬WiFi is stored in the cache. The class concept ¬UserRequest is also stored in the cache because one of its sub-conditions generated a clash (i.e. a match). These concepts are stored as follows. Let t denote the current date/time. Firstly, the clashing concept ¬WiFi, which has no dependants, is stored as $\langle$¬WiFi, $null$, t, $true\rangle$. The last status field is set to $true$ because ¬WiFi generated a clash for all possible expansions (there was only one possible expansion for ¬WiFi). Additionally, ¬WiFi depends on ∀ hasComm.¬WiFi (which has only one possible expansion) leading to the tuple $\langle$∀ hasComm.¬WiFi, ¬WiFi, t, $true\rangle$ being added to the cache. ∀ hasComm.¬WiFi depends on ¬InternetRequest which results in the tuple $\langle$¬InternetRequest, ∀ hasComm.¬WiFi, t, $false\rangle$ being added to the cache. This time the status field was set to $false$ indicating a partial match, because ¬InternetRequest is a disjunction which has two possible expansions and only one has been found to clash so far. Finally, ¬InternetRequest depends on ¬UserRequest which results in the tuple $\langle$¬UserRequest, ¬InternetRequest, t, $false\rangle$ being added to the cache. Again the status is $false$ because ¬InternetRequest has only partially clashed.

Now that we have defined the structure of our cache, we will detail storage and retrieval from our cache in the next section.

4.5.3 Cache Storage and Retrieval

New tuples are added to the cache whenever a clash occurs, during the inference checking process. More specifically, when a clash is generated, the assertion which has given rise to the clash is stored in the cache, as well as all the assertions which it depends on. Cache retrieval occurs whenever a new

disjunction element is applied during the inference checking process. If the disjunction element being applied is found in the cache, and this element has generated a full clash, then we know that this request condition has generated a clash for all subsequent expansions in a past request. As long as the cache entry was added after a user specified time constraint, then the reasoner immediately generates a clash without having to apply all of the expansions again. Alternatively, if the cache entry was added before the user specified expiry time, or did not generate a full clash (for all expansions), then all assertions which are dependant on the entry, are retrieved, and transformation rules are applied to these assertions in priority to all other assertions. Now we will first describe cache storage, followed by cache retrieval.

Cache Storage

As mentioned previously in this section, the cache is a set of tuples. The cache is serialised and stored to file on persistent secondary storage such as an SD card on a mobile device. The cache is loaded into main memory whenever an inference checking process begins, and stored back to secondary storage whenever the inference check is completed.

Algorithm 4.3 defines the process involved for adding new tuples to the queue when a clash occurs. During the inference checking process, a clash may be generated when a transformation rule is applied to an assertion $C(x)$. When a clash is generated, a call is made to Algorithm 4.3 where the clashing assertion $C(x)$ is passed as the first parameter and a value of *true* is passed as the second parameter *status* indicating that this is the first call to the algorithm (in subsequent recursive calls *status* is $false$).

The main functionality of the algorithm is to store the assertion which has generated the clash to the cache, as well as to store all assertions which the clash generating assertion depends on as separate entries in the cache. This occurs using recursive call backs. The algorithm makes use of the $AddCacheEntry$

Algorithm 4.3 ClashDetected$\Big(C_j(x_p),\ status\Big)$

Inputs: Assertion $C_j(x_p)$, Boolean *status*

1: **if** $status =$ **true then**
2: $\quad AddCacheEntry\Big(C_j(x_p), \textbf{null}, \textbf{true}\Big)$
3: $\quad status \leftarrow$ **false**
4: **end if**
5: Let $C_i(x_r) \leftarrow dependsOn\Big(C_j(x_p)\Big)$
6: **if** $C_i(x_r) \neq$ **null then**
7: $\quad$ Let $fullClash \leftarrow$ **false**
8: $\quad$ **if** $\Big(C_i(x_r) = C_1 \sqcup ... \sqcup C_n(x_r), 1 \leq k \leq n\Big)$ and $\Big(s = true$, for all $\langle C_k(x_r), \alpha', s\rangle \in CSCache$, for any $\alpha'\Big)$ **then**
9: $\quad\quad fullClash \leftarrow$ **true** *$C_j(x_p)$ depends on an assertion $C_i(x_r)$ where C_i is a disjunction and all of its disjunct element assertions $C_k(x_r)$ have been found to clash*\
10: $\quad$ **end if**
11: $\quad AddCacheEntry\Big(C_i(x_r), C_j(x_p), fullClash\Big)$
12: $\quad ClashDetected\Big(C_i(x_r), status\Big)$
13: **end if**

algorithm, which is defined in Algorithm 4.4, to actually add an entry to the cache. Thus, Algorithm 4.3 first adds to the cache, the assertion which generated the obvious clash when a transformation rule was applied to it. The algorithm then obtains the assertion $C_i(x_r)$ which $C_j(x_p)$ depends on. Let $dependsOn\Big(C_j(x_p)\Big)$ denote the assertion which $C_j(x_p)$ depends on. If C_i is a disjunction, then the $fullClash$ attribute is set to $true$ if all of the disjunct elements of the disjunction C_i are contained in the cache and marked as full matches. If all elements of the disjunction C_i are not marked as full clashes then this implies that there is at least one possible expansion for C_i which has not yet been found to generate a clash, which means our proposed CS strategy cannot avoid evaluating $C_i(x_r)$. The assertions $C_i(x_r)$ and $C_j(x_p)$ are added to the cache using a call to *AddCacheEntry*.

Algorithm 4.4 adds a new entry to the cache. That algorithm is invoked using the call $AddCacheEntry\Big(C_i(x_r), C_j(x_p), s\Big)$, where $C_i(x_r)$ and $C_j(x_p)$ are assertions such that $C_i(x_r)$ depends on $C_j(x_p)$ and s is the status field which is set to $true$ if entry is for a full clash (i.e. a full match) or $false$ otherwise (i.e. a partial match). If there is already an entry in the cache the pair $C_i(x_r)$

Algorithm 4.4 AddCacheEntry$\Big(C_i(x_r),\, C_j(x_p),\, s\Big)$

Inputs: Assertion $C_i(x_r)$, Assertion $C_j(x_p)$, Boolean s

1: remove $\Big\langle C_i(x_r), C_j(x_p), t', s' \Big\rangle$ from $CSCache$ for any t', s'
2: **if** reached memory limit for $CSCache$ **then**
3: remove all $\langle \alpha', \alpha'', t'', s' \rangle \in CSCache$, for any α', α'', s' where t'' is the oldest date/time stamp in $CSCache$
4: **end if**
5: Let $t \leftarrow$ current date/time stamp
6: $CSCache \leftarrow CSCache \cup \Big\langle C_i(x_r), C_j(x_p), t, s \Big\rangle$

and $C_j(x_p)$ then this is removed, because the existing entry will be replaced by the new entry. In addition, since small devices have varying amounts of available storage, it is possible that the cache might be full. If this is the case, the entries which share the oldest time stamp are removed. Finally the new entry is added to the cache.

Cache Retrieval

As described in Section 4.5.1, a cache entry about an assertion contains a marker to signify whether or not it represents a full or partial clash. In the case that it is marked as a full clash then it can be used to immediately generate a clash in the reasoner, without further evaluating the assertion by applying transformation rules. This is provided the time stamp associated with the cache entry indicates that it falls within any user specified age limit. This implies that for an assertion $C_i(x_r)$, there is an entry in the cache $\langle C_i(x_r), \alpha, t, s \rangle$, where t must be greater than the user specified age limit / expiry time, s must be *true* and α can be any value. This will cause an immediate clash for $C_i(x_r)$. If this occurs, then the reasoner will not evaluate $C_i(x_r)$ any further.

Alternatively, if the entry is marked as a partial clash, or the user specified age limit was not met, the assertion $C_i(x_r)$ must be fully evaluated by the reasoner. The remainder of the subsections describing the caching strategy are concerned with these assertions which are contained in the cache, but must be re-evaluated. Priority will be given to any assertions which depend

on $C_i(x_r)$ and generated a clash previously, to improve response time. These assertions are retrieved from the cache and associated with a weight value to indicate this increased priority. The weight value is calculated when the cache entry is retrieved only. It is not stored in the cache because it would require updating whenever entries are added or removed from the cache. In addition, it is influenced by which assertion $C_i(x_r)$ is being retrieved from the cache. For instance, if retrieved assertion $C_i(x_r)$ represents a full clash in the cache it is given a weight value of 1 and the weight value of all dependant assertions form a proportion of this value. Let $weight\Big(C_i(x_r)\Big)$ be the weight value associated with an assertion $C_i(x_r)$, where $1 \leq weight\Big(C_i(x_r)\Big) \leq 0$. The assertions retrieved from the cache and their associated weight values form pairs as defined in Equation 4.3. It is noteworthy to mention that all assertions in the reasoner which are not found in a cache entry have a default weight of zero.

$$\left\{\left\langle C_1(x_1), weight\Big(C_1(x_1)\Big)\right\rangle, ..., \left\langle C_n(x_m), weight\Big(C_n(x_m)\Big)\right\rangle\right\} \quad \text{where}$$
$$1 \leq i \leq n, \quad 1 \leq r \leq m \tag{4.3}$$

The process of retrieving the set of assertions and weight value pairs from the cache, is specified in Algorithm 4.5. This algorithm retrieves all dependant assertion and weight value pairs for the assertion $C_i(x_r)$ and returns a set $\mathcal{P}$ as an output, which contains dependant assertion and weight value pairs. The algorithm is invoked using the call $GetDependants\Big(C_i(x_r), 1\Big)$, where $C_i(x_r)$ is the disjunction element being applied by the reasoner and 1 is the highest weight value which should be associated with an assertion if all of its possible expansions were found to generate a contradiction previously. This second parameter u is always initially 1.

Algorithm 4.5 GetDependants($C_i(x_r)$, u)

Inputs: Assertion $C_i(x_r)$, double u, where x_r is an individual which is a member of the class concept C_i

Outputs: $\langle$Set $\mathcal{P}$, double $w\rangle$

1: Let $\mathcal{P} \leftarrow \emptyset$
2: Let D be a set of assertions $C_j(x_p)$, for all $\Big\langle C_i(x_r), C_j(x_p), t, s\Big\rangle \in CSCache$ where t and s can be any value
3: Let $w \leftarrow 0$
4: **if** $D \neq \emptyset$ **then**
5: **for all** $C_j(x_p) \in D$ **do**
6: **if** $C_i \equiv C_1 \sqcup ... \sqcup C_n, 1 \leq j \leq n$ **then**
7: $u \leftarrow u/n$ *all C_j are disjunct elements of C_i, where $C_j(x_p) \in D$*\
8: **end if**
9: Let $weight\Big(C_j(x_p)\Big) \leftarrow u$
10: $\mathcal{P} \leftarrow \mathcal{P} \cup \left\{\left\langle C_j(x_p), weight\Big(C_j(x_p)\Big)\right\rangle\right\}$
11: Let $\langle \mathcal{P}', w'\rangle \leftarrow GetDependants\Big(C_j(x_p), u\Big)$
12: $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}'$
13: **if** $C_i \equiv C_1 \sqcup ... \sqcup C_n, 1 \leq j \leq n$ **then**
14: $w \leftarrow w + w'$ *sum dependant disjunct weights*\
15: **else if** $w' > w$ **then**
16: $w \leftarrow w'$ *take highest weight of dependants*\
17: **end if**
18: **end for**
19: **else**
20: **return** $\langle \mathcal{P}, u\rangle$ *no dependants*\
21: **end if**
22: **return** $\langle \mathcal{P}, w\rangle$

The algorithm, obtains from the cache, all the assertions $C_j(x_p)$ which are dependant on $C_i(x_r)$, where $C_i(x_r)$ is passed as a parameter to the algorithm. Assertions which depend on $C_i(x_r)$, are retrieved by getting all tuples from the cache which contain $C_i(x_r)$ in the first field and storing all dependants $C_j(x_p)$ into a set D, where $C_j(x_p)$ is an element contained in the second field of the tuple where $C_i(x_r)$ is the first field. Thus, D is the set of obtained dependants for $C_i(x_r)$. The algorithm then loops each of these assertions $C_j(x_p) \in D$. It allocates weight values to each $C_j(x_p)$, where a weight value indicates the normalised number of possible expansions of $C_j(x_p)$ which are stored in the cache as generating a full clash. Normalised implies that all values are between 0 and 1. If C_i is a disjunction then any assertions which directly

depend on $C_i(x_r)$ will represent disjunct elements of C_i. In this case, the current weight value u is divided by the number of disjunct element members which the disjunction C_i, contains (whether or not these are stored in the cache as dependants for C_i). This is because each disjunct element represents an alternative expansion which depends on C_i and all alternative expansions must generate a full clash in order to prove a full clash (i.e. a match) for an assertion $C_i(x_r)$ about the disjunction C_i. In the algorithm, the weight of the current $C_j(x_p)$ is set to be the current weight value u.

Each $C_j(x_p)$ and its weight value pair are added to the set $\mathcal{P}$, which will be returned as an output by Algorithm 4.5. The current $C_j(x_p)$ and normalised weight value u is then passed to $GetDependants$, which is invoked using a recursive call. This will recursively retrieve the dependants of each $C_j(x_p)$ in the same way until all dependants have been evaluated. This call to $GetDependants$ will return a new set $\mathcal{P}'$ of assertion and weight value pairs, and a value w' which is the highest weight value associated with an assertion in $\mathcal{P}'$. $\mathcal{P}'$ is added to the current set $\mathcal{P}$. If the current class concept C_i is a disjunctive assertion, this means all dependant assertions $C_j(x_p)$ relate to disjunctive elements C_j, so each w' is summed and the total w is used. Otherwise w is the highest weight value w' of those weights associated with the dependants $C_j(x_p)$. We take the highest weight to cater for conjunctions, for which there may be multiple dependants, each representing a conjunct element, however only one of these needs to generate a full clash to prove a full clash for the conjunction to which it is a member of. Finally, after looping all dependant assertions of $C_i(x_r)$ in the cache, the set $\mathcal{P}$ and value w are returned as outputs for the algorithm.

During the inference checking process, the weight values $weight\Big(C_i(x_r)\Big)$ will be used to determine the priority for which assertion $C_i(x_r)$ to evaluate next. Current Tableaux reasoners employ an unordered $ToDo$ list of assertions to evaluate (i.e. by applying transformation rules to them) as described in

Section 3.4.2. We replace the $ToDo$ list with queues which order assertions by weight value. It is noteworthy that the value w is only used by $GetDependants$ to hold the weight value of the assertion which the parameter $C_i(x_r)$ depends on, to manage the recursive calls. It is not used by the reasoner for the matching process.

Our priority queues which make use of weight values $weight\Big(C_i(x_r)\Big)$ for assertions $C_i(x_r)$ will be discussed in the next section.

4.5.4 CS Weighted Queues

In the case that an assertion is obtained from the cache, as having generated a clash in the past, but has either generated a partial clash or the assertion was stored in the cache before the user specified age limit, then this assertion must be evaluated before the others in order to improve response time. In the previous section we described how assertions are obtained from the cache and allocated with weight values. These weight values are used to indicate a priority level. Assertions with the highest priority are evaluated first. Any assertion which was not found in the cache, has a priority weight value of zero, meaning these will be evaluated last. Thus, we establish weighted queues for ordering assertions by weight value. These queues replace the unordered $ToDo$ list of assertions which is employed by current reasoners.

The $ToDo$ list, which current reasoners employ, contains all assertions which can have transformation rules applied to them. However, as discussed previously, the $\sqcup$-rule, which is applied to disjunctions, is a non-deterministic rule. This means it gives rise to many possible expansions which must each be checked in order to prove every possible expansion generates a contradiction. This significantly increases the size of the search space. Therefore, in our CS strategy, we have two queues. One queue contains disjunctive assertions and the second queue contains all the other assertions. Let $\mathcal{Q}^d$ be the weighted queue which contains disjuntive assertions and let $\mathcal{Q}^o$ be the weighted queue

which contains any other assertion. The main idea is that we apply assertions from $\mathcal{Q}^o$, then we apply only a single disjunction in $\mathcal{Q}^d$, which may add more assertions to $\mathcal{Q}^o$. We continue this process, until all assertions are applied.

The weight value $weight\Big(C(x)\Big)$ of any assertion $C(x)$ begins at zero. Weight values might then be increased by Algorithm 4.5 which retrieves assertions from the cache, as described in the previous section. Assertions are added to their respective queue, whenever these are added to the ABox $\mathcal{A}$. This may occur when an ontology is loaded into the ABox $\mathcal{A}$, or when a transformation rule adds additional assertions to the ABox $\mathcal{A}$. However, as stated in Section 3.4.2 the $\forall$-rule and $\sqcap$-rule are applied to assertions containing universal quantifiers or conjunctions as soon as these are added to the ABox $\mathcal{A}$. As a result, these assertions are not added to the $ToDo$ list by current reasoners and are, therefore, not added to our queues either. Both queues $\mathcal{Q}^o$ and $\mathcal{Q}^d$ are ordered in descending $weight\Big(C(x)\Big)$ order which is defined in Equation 4.4.

$$\mathcal{Q} = \left\{ \Big\langle C_1(x_1), weight\Big(C_1(x_1)\Big) \Big\rangle, ..., \Big\langle C_n(x_m), weight\Big(C_n(x_m)\Big) \Big\rangle \right\} \quad \text{where}$$
$$weight\Big(C_i(x_r)\Big) \geq weight\Big(C_j(x_p)\Big),\ 1 \leq i \leq j \leq n,\ \ 1 \leq r \leq p \leq m \tag{4.4}$$

The considerations discussed in this section with regard to assertion reordering using our queues, is illustrated more formally in Algorithm 4.6. This algorithm is given the ABox, expansion tree and the two queues as input parameters. It uses the queues to determine which assertion to apply next and applies the appropriate transformation rule to it. If the application of a transformation rule causes a clash then the class concept which has generated the contradiction is returned as output. Let $ApplyTransformationRule$[1] denote an algorithm which applies the appropriate standard Tableaux transformation

[1]The $ApplyTransformationRule$ function applies the standard Tableaux transformation rules as outlined in Section 3.4.2 and is therefore not shown

rule the single assertion which is provided to it as an input parameter and returns a class concept if a clash was generated or *null* otherwise. It is noteworthy that these transformation rules will also encompass the optimisation strategies defined in sections 4.3 and 4.4 if these are enabled.

Algorithm 4.6 ApplyTableauxRules($\mathcal{A}$, $\mathcal{G}$, $\mathcal{Q}^o$, $\mathcal{Q}^d$)

Inputs: ABox $\mathcal{A}$, ExpansionTree $\mathcal{G}$, Queue $\mathcal{Q}^o$, Queue $\mathcal{Q}^d$
Outputs: Boolean *clashDetected*

1: Let $C_i(x_r)$ be the assertion in $\mathcal{Q}^d$ with the highest $weight\Big(C_i(x_r)\Big)$
2: **while** $\mathcal{Q}^o \neq \emptyset$ **do**
3: Let $C_j(x_p)$ be the assertion in $\mathcal{Q}^o$ with the highest $weight\Big(C_j(x_p)\Big)$
4: **if** $weight\Big(C_j(x_p)\Big) = 0$ and
$\Big(C_i(x_r) \neq$ **null** and $weight\Big(C_i(x_r)\Big) > 0\Big)$ **then**
5: **break** out of while loop *apply next disjunction instead*\
6: **end if**
7: remove $\Big\langle C_j(x_p), weight\Big(C_j(x_p)\Big)\Big\rangle$ from $\mathcal{Q}^o$
8: Let $clash \leftarrow ApplyTransformationRule\Big(C_j(x_p), \mathcal{A}, \mathcal{G}\Big)$
9: **if** $clash \neq null$ **then**
10: **return** *clash*
11: **end if**
12: **end while**
13: **if** $\mathcal{Q}^d \neq \emptyset$ and $C_i(x_r) \neq$ **null** **then**
14: remove $\Big\langle C_i(x_r), weight\Big(C_i(x_r)\Big)\Big\rangle$ from $\mathcal{Q}^d$
15: Let $clash \leftarrow ApplyTransformationRule\Big(C_i(x_r), \mathcal{A}, \mathcal{G}\Big)$
16: **end if**
17: **return** *clash*

Algorithm 4.6 applies all assertions in $\mathcal{Q}^o$ which have a weight higher than zero, until a clash occurs. If there are no more assertions in $\mathcal{Q}^o$ with a weight higher than zero, but there is a disjunctive assertion in $\mathcal{Q}^d$ with a weight higher than zero, then one disjunctive assertion is applied. The *ApplyTableauxRules* algorithm will be called by the inference checking process until there are no more transformation rules to apply or there is a clash for every expansion.

More specifically, the algorithm loops while the queue $\mathcal{Q}^o$ is not empty. It retrieves the assertion $C_j(x_p)$ from $\mathcal{Q}^o$ which has the highest weight value $weight\Big(C_j(x_p)\Big)$ in the queue. If this assertion $C_j(x_p)$ has a weight which is higher than zero, then it removes $C_j(x_p)$ from the queue $\mathcal{Q}^o$ and applies the

appropriate Tableaux transformation rule to it. If applying the transformation rule to the assertion $C_j(x_p)$ generates a clash, then the algorithm stops and returns the clashing concept. In the case that the next $C_j(x_p)$ from $\mathcal{Q}^o$ has a weight of zero, but the disjunction queue $\mathcal{Q}^d$ contains assertions which have a weight higher than zero, then the algorithm breaks out of the loop to apply the next disjunctive assertion instead. As such, if the disjunction queue $\mathcal{Q}^d$ is not empty, the disjunctive assertion $C_i(x_r)$ with the highest weight value $weight\Big(C_i(x_r)\Big)$ is retrieved from the queue $\mathcal{Q}^d$ and removed. The $\sqcup$-rule is applied to $C_i(x_r)$. If applying the transformation rule to the assertion $C_i(x_r)$ generates a clash, the clashing concept is returned.

As described in Section 3.4.2 previously, whenever an assertion $C(x)$ is added to the $ToDo$ list, it is associated with the current branch point node b_{i+1} in $\mathcal{G}$, known as a dependency. When the reasoner backjumps to an earlier branch point node b_i, an assertion $C(x) \in ToDo$ is removed from the $ToDo$ list if $C(x)$ depends on a branch node b_{i+1} (i.e. b_{i+1} was added at or after b_i). Since our queues $\mathcal{Q}^o$ and $\mathcal{Q}^d$, which were described in this section, replace the $ToDo$ list, the same dependencies and backjumping functionality applies to our queues $\mathcal{Q}^o$ and $\mathcal{Q}^d$. As such, when the reasoner backjumps to an earlier branch point node b_i an assertion $C(x) \in \mathcal{Q}^o$ or $C(x) \in \mathcal{Q}^d$ is removed from its queue if $C(x)$ depends on a branch node b_{i+1} (i.e. b_{i+1} was added at or after b_i). In the next section we illustrate our caching strategy (CS) using an example.

4.5.5 Example

When our proposed caching strategy (CS) is enabled, all assertions which generate clashes and those assertions which these depend on, are stored in the cache. Assume that the user performs the inference check where the service description x_3 was matched against the user request definition $\exists R_1.(C_2 \sqcap C_3)$. Assume the inference checking process was executed at 10am on 2009-10-01.

The inference checking process involves asserting the negation of C_3 as a type for x_3, where $\neg C_3 \equiv \forall R_1.(\neg C_2 \sqcup \neg C_3)$, on the ABox from the previous example in Section 3.4.3. This request is illustrated in Figure 4.6 and it comprises a sub-part of the inference example from Section 3.4.3 using standard Tableaux. In the figure, the class concept $\neg C_2$ which is asserted as a type for the individual x_4 generates a clash and the concepts $\neg C_2$ and $\neg C_3$ which are asserted as a types for the individual x_5 both generate a clash. These assertions and the assertions which these depend on, are stored in the cache. The resulting cache entries are illustrated in figure 4.7. The nodes shaded in yellow indicate transformations which contributed to a full clash in the cache.

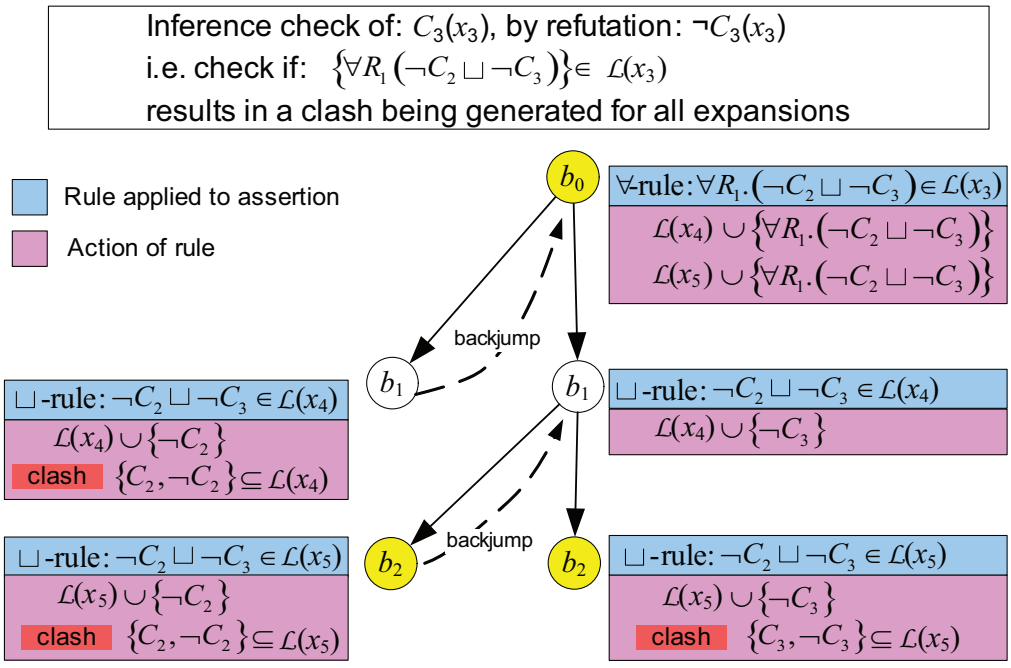


Figure 4.6: First Example CS Inference Check

Assume now that the user request from the standard Tableaux example in Section 3.4.3, is being performed with the proposed CS caching strategy enabled. This example checks whether the class concept C_0 definition can be inferred to be a type for the individual x_0, by asserting the negation of C_0 as a type for x_0, where $\neg C_0 \equiv \forall R_2.(\leq 0 R_2) \sqcup \forall R_1.(\neg C_1 \sqcup \forall R_1.(\neg C_2 \sqcup \neg C_3))$. Assume that there is no user specified age limit on the request conditions

CSCache

$$\Big\{\Big\langle \forall R_1.(\neg C_2 \sqcup \neg C_3)(x_3), \neg C_2 \sqcup \neg C_3(x_4), \text{2009-10-01T10:00}, false\Big\rangle,$$
$$\Big\langle \forall R_1.(\neg C_2 \sqcup \neg C_3)(x_3), \neg C_2 \sqcup \neg C_3(x_5), \text{2009-10-01T10:00}, true\Big\rangle,$$
$$\Big\langle \neg C_2 \sqcup \neg C_3(x_4), \neg C_2(x_4), \text{2009-10-01T10:00}, false\Big\rangle,$$
$$\Big\langle \neg C_2(x_4), null, \text{2009-10-01T10:00}, true\Big\rangle,$$
$$\Big\langle \neg C_2 \sqcup \neg C_3(x_5), C_2(x_5), \text{2009-10-01T10:00}, true\Big\rangle,$$
$$\Big\langle \neg C_2 \sqcup \neg C_3(x_5), C_3(x_5), \text{2009-10-01T10:00}, true\Big\rangle,$$
$$\Big\langle \neg C_2(x_5), null, \text{2009-10-01T10:00}, true\Big\rangle,$$
$$\Big\langle \neg C_3(x_5), null, \text{2009-10-01T10:00}, true\Big\rangle\Big\}$$

Figure 4.7: Example Caching Strategy (CS) Cache Entries

retrieved from the cache. Figure 4.8 presents the same Tableaux expansion process as in 3.4.3, except that when the reasoner attempts to apply the definition $\forall R_1.(\neg C_2 \sqcup \neg C_2)(x_3)$ it detects that this has generated a clash for all expansions (i.e. a full clash) previously. Thus, the caching strategy eliminates the application of transformation rules to $\forall R_1.(\neg C_2 \sqcup \neg C_2)(x_3)$ and its dependants. The eliminated transformations have a red line through them in the figure, while the transformations which were applied are highlighted in yellow.

Alternatively, assume the user does specify an age restriction stating that the request condition $\exists R_1.(C_2 \sqcup C_3)$ being matched against x_3, must have been stored in the cache after 11am on 2009-10-01, otherwise it needs to be re-evaluated. The negated condition $\forall R_1.(\neg C_2 \sqcup \neg C_3)(x_3)$ is stored in the cache, but it does not meet this time stamp expiry. Therefore, assertions which depend on $\forall R_1.(\neg C_2 \sqcup \neg C_3)(x_3)$ are retrieved from the cache, and associated with weight values in the queues, indicating their level of priority.

This gives rise to the queue of disjunctions $\mathcal{Q}^d$ and the queue of all other assertions $\mathcal{Q}^o$ shown in Figure 4.9. The $\forall R_1.(\neg C_2 \sqcup \neg C_3)$ definition asserted

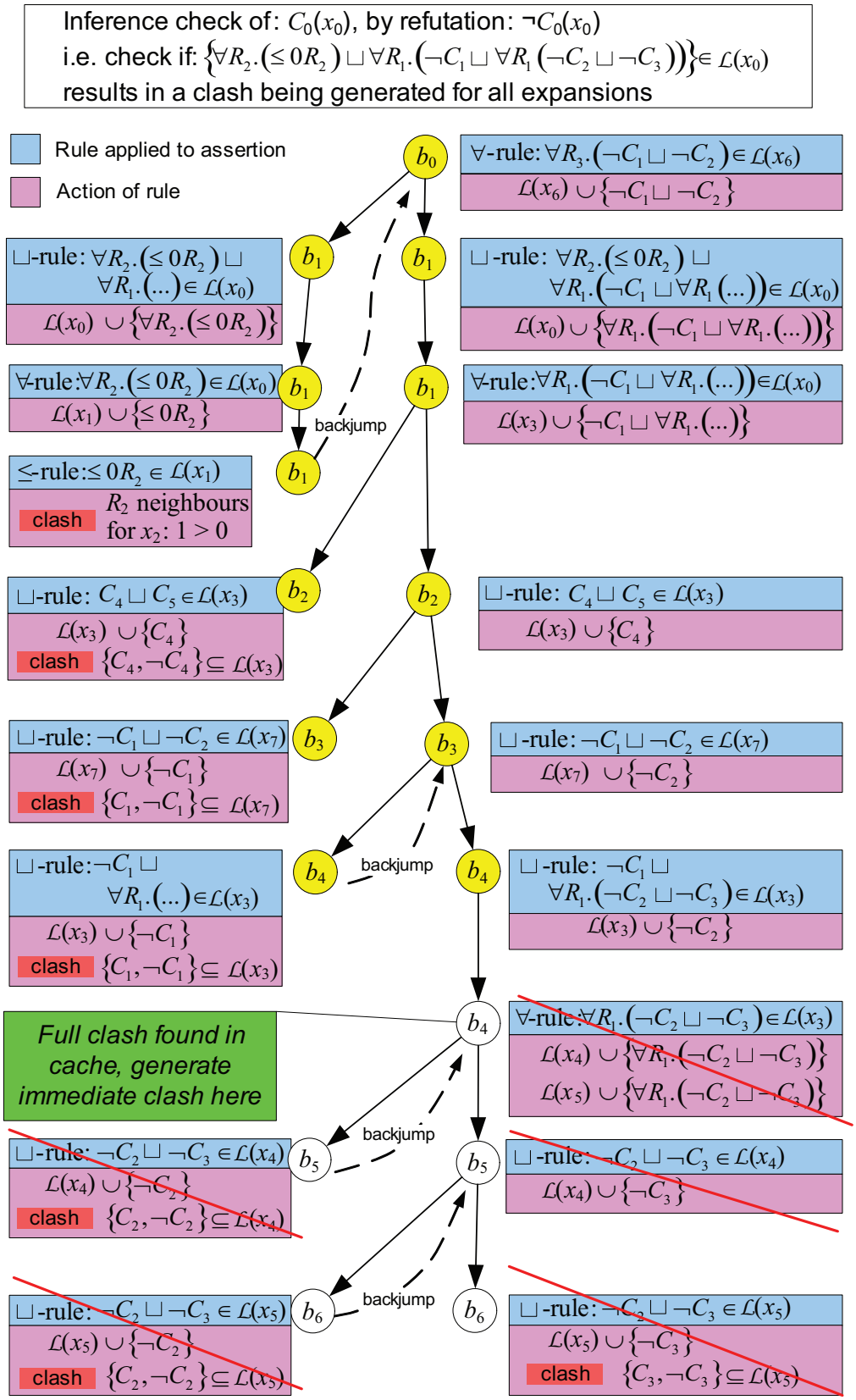


Figure 4.8: Second Example CS Inference Check

to x_3 is applied first. This adds the disjunction $\neg C_2 \sqcup \neg C_3$ to x_4 with a weight of 0.5 (i.e. partial clash) and to x_5 with has a weight of 1.0 (i.e. full clash).

Queue $\mathcal{Q}^d$	Weight
$\neg C_2 \sqcup \neg C_3(x_5)$	1.0
$\neg C_2 \sqcup \neg C_3(x_4)$,	0.5

Queue $\mathcal{Q}^o$	Weight
$\forall R_1.(\neg C_2 \sqcup \neg C_3)(x_3)$	1

Figure 4.9: Example Caching Strategy (CS) Queues

Using this queue gives rise to the application of transformation rules which are illustrated in figure 4.10. This is identical to the inference check from Section 3.4.3 except that the disjunction $\neg C_2 \sqcup \neg C_3$ which is added as a type to x_5, is applied before the same disjunction which is added to x_4. This is a re-ordering of transformation rule application. Since $\neg C_2 \sqcup \neg C_3(x_5)$ clashes for all expansions, there is no need to evaluate the disjunction added to x_4, because the inference is proven. The eliminated transformations are shown in the figure as having a red line through them.

As such, it can be seen from this example, even if the user specified age limit is exceeded, the cache can still significantly reduce the number of transformation rules applied. Therefore, as illustrated in the example presented in this section, we hypothesise that our CS caching strategy will lead to a significant performance gain when cache entries generate an immediate clash during an inference check and will also lead to gains in response time when cache entries are used to re-order transformation rule application, without compromising accuracy.

4.6 Summary

In this chapter we presented our optimisation and caching strategies which we call mobile Tableaux (mTableaux) to enable on-board mobile inference / reasoning, which form a key contribution of our thesis. Our strategies include:

1. Selective Application of Tableaux Transformation Rules (ST);

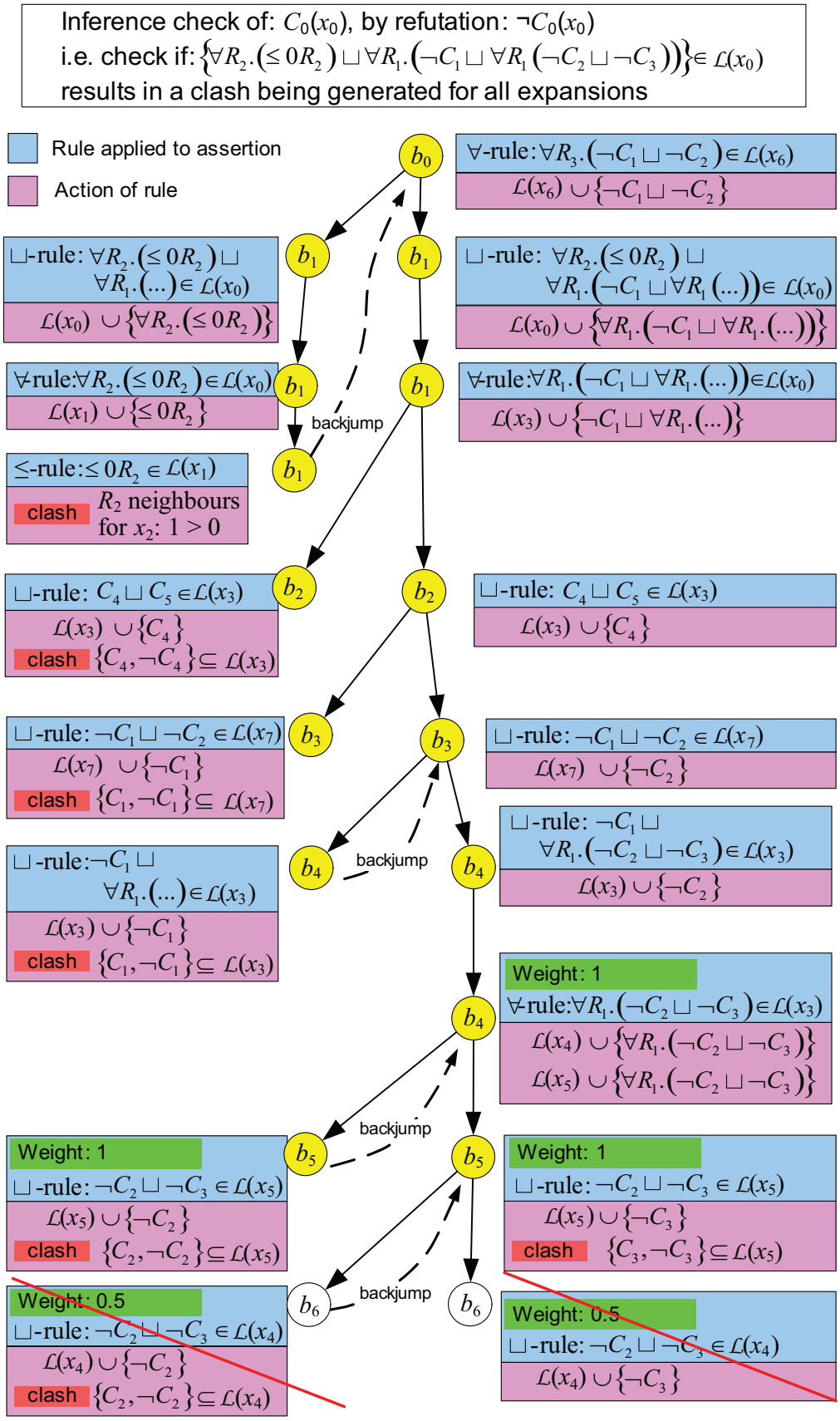


Figure 4.10: Second Example CS Inference Check: Re-evaluate Expansions

2. Selective Disjunction Transformation Rule Application (SD);

3. Caching Strategy (CS).

The ST and SD strategies limit the number of Tableaux transformation rules which are applied to reduce the size of the reasoning task, without significantly compromising result accuracy. The ST strategy applies transformation rules to a subset of individuals only. These individuals are selected based on the service description being checked. Additionally, the SD strategy limits the number of disjunctions applied, since disjunctions generate Tableaux tree expansions, giving rise to many branch possibilities to evaluate. A disjunction is only evaluated if it contains a class concept which relates to the user request. Furthermore, CS caches the evaluations which gave rise to match findings for previous request to service description comparisons. The cache is used to reduce the response time of subsequent comparisons which are similar to those performed previously. Depending on the level of completeness required for particular conditions in the user request, a cache entry can be used to generate an immediate match, or it can be used to re-order the evaluations completed by the Tableaux algorithm, so that those which contributed to a match finding previously, are applied first. We hypothesis that our optimisation and caching strategies will lead to a significant improvement in response time. We will present a comprehensive performance evaluation to validate this hypothesis in Chapter 6.

In addition, to cater for resource and user requirements, we provide an adaptive inference strategy which enables priority based matching of the requirements in the user request. Our strategy can be interrupted at any stage during the matching process depending on user constraints such as available time or computational resources, to provide a match result based on the processing completed up to the point of interruption. This forms another key contribution of our thesis, and will be outlined in the next chapter.

Chapter 5

Adaptive Strategies for Mobile Inference

5.1 Introduction

In the previous chapter we outlined our approach to light-weight inference to enable accurate and efficient mobile matching of service descriptions with user requests. This involved the proposal and development of optimisation and caching strategies for the Tableaux decision proof. In this chapter we extend our light-weight strategies to take account of user and resource constraints, by proposing an adaptive inference strategy. Our proposed strategy supports "anytime" (Wache et al., 2004) matching with an innovative capacity to provide incremental / partial matching of a request to a service description.

The Tableaux expansion procedure, which we detailed in Section 3.5, is widely used as the proof algorithm for current reasoners (Baader et al., 2003, p. p. 322). In this Tableaux procedure, expansion occurs in a depth-first order with no particular assessment of priorities for the expansion evaluations. In addition, while partial matching have been employed by semantic matching approaches (Skoutas et al., 2007; Lu, 2005; Srinivasan et al., 2005), semantic reasoners do not support this functionality. Current Tableaux reasoners

require that the matching task is completed in full before the match result is given, otherwise no result is provided. Thus they operate on an "all or nothing" principle. Furthermore, if any condition in the user request fails to match the service description, a negative inference match result is given, even if the requirement was not a significant feature for the user. We see these characteristics of current semantic reasoners (that may not have any bearing on suitability for desktop environments) provide an opportunity for further optimisation of performance in resource constrained environments. Therefore, in our adaptive inference strategy, we propose modifications to the Tableaux algorithm to enable:

- prioritised matching of requirements in the user request based on importance of these to the user;
- *anytime inference* in which the reasoner can be stopped prematurely depending on constraints such as resource or time constraints, thereby providing incremental results;
- support for a metric that specifies a weighted degree of match which indicates the strength of the match based on the extent of the evaluation / processing actually completed, thereby supporting partial degree of matching.

The contributions in this chapter have led to the following research publications: (Steller et al., 2009b,a).

The remainder of this chapter is structured as follows. In Section 5.2 we will review other related research in the area of incremental, priority based, "anytime" reasoning. In Section 5.3 we will provide an overview of our resource-aware adaptive strategy and our proposed modifications to the Tableaux algorithm. This section will also contain an illustrative example and complexity comparison between standard Tableaux and our proposed strategy. In Section 5.4 we detail the notation and representation of weighted disjunctions and their

dependencies which will be used throughout this chapter. In Section 5.5 we will discuss the association of user specified weight values with conditions in the user request. These are used to generate additional weight values used by our proposed strategy. In Section 5.6 we will detail the way the next disjunction to expand is selected, which involves queues and priority based algorithms. In Section 5.7 we discuss the way our adaptive strategy maintains state. Finally in Section 5.8 we discuss the way in which a degree of match value is calculated by our strategy.

5.2 Adaptive Reasoning Approach

Current reasoners such as Pellet[1], RacerPro[2], FaCT++[3] and KAON2[4] are considerably resource intensive (Fensel et al., 2008) and do not scale to small resource constrained devices. This is rooted in underlying assumptions of current systems for logic reasoning. Current reasoners assume completeness and correctness of the knowledge which is being reasoned upon and emphasise completeness and correctness in the inferences made about this knowledge. However, in dynamic and distributed mobile environments this focus on complete and correct knowledge brings significant computational overhead. Thus, while accuracy of inference results are important, performance efficiency should also be taken into consideration by reasoners. In this thesis, we develop strategies for mobile semantic reasoning which aim to balance the trade-off between completeness of inference reasoning with acceptable levels of accuracy and computational performance. In this section we will provide a review of current research in the areas of fuzzy logic and approximation, which is taken into consideration by our approach.

[1]http://clarkparsia.com/pellet/ (accessed May 2009)
[2]http://www.racer-systems.com/ (accessed May 2009)
[3]http://owl.man.ac.uk/factplusplus/ (accessed May 2009)
[4]http://kaon2.semanticweb.org/ (accessed May 2009)

Fuzzy logic[5] (Zadeh, 1965, 1975; Konar, 2005) is a technique which deals with imprecise reasoning rather than traditional crisp binary logic. Generally fuzzy logic is used to reason under uncertainty, where uncertainty values can be associated with definitions in Description Logic and reasoning can provide a strength of membership result between zero and one (Lukasiewicza and Straccia, 2009; Stoilos et al., 2008; Li et al., 2006; Giugno and Lukasiwicz, 2002; Straccia, 2005). Ragone et al. (2008), and Kuster and König-Ries (2008) employ a fuzzy approach to support imprecise / partial matching of a user request against a service description. The main drawback of these approaches is that they do not perform matching of the conditions in the user request in priority order. In addition, they do not support an incremental reasoning approach which can be interrupted prematurely based on user constraints such as resource and time limits.

There is an emerging body of research which does attempt to address the need for incremental reasoning which is known as approximate reasoning (Rudolph et al., 2008). Under this approach a subset of the user request is compared against the service description and the match result is said to be an approximation of the full user request. The size of those subset is incrementally increased and compared with the user request until the full request has been checked. The process can be stopped at any increment and is, thus, considered to be "anytime" (Wache et al., 2004). This avoids the need to match a request with a service description in full (Fensel and van Harmelen, 2007), before providing a result and trades accuracy with efficiency (Fensel et al., 2008).

There are a number of approaches which support approximate reasoning with Description Logic which can be used for matching user requests with semantically described service descriptions, using current reasoners which generally employ the Tableuax algorithm (Stuckenschmidt and Harmelen, 2002;

[5]http://plato.stanford.edu/entries/logic-fuzzy/ (accessed 18-01-2010)

Stuckenschmidt and Kolb, 2008; Schlobach et al., 2007; Wache et al., 2005). There are also approximate reasoning techniques designed specifically for the Datalog inference algorithm used by the KOAN2 reasoner[6] (Rudolph et al., 2007; Hitzler and Vrandecic, 2005). However, these approaches still provide only a binary result indicating a match or no match and do not support priority ordering of conditions in the user request based on the importance of these to the mobile user. Iranmanesh et al. (2009) addresses this by associating priority values with conditions in the request, based on both heuristics and user specified importance values. These can be used for determining the order in which queries are executed and to provide a degree of match value when reasoning is stopped.

The main feature of these approaches is that they successively compare subsets of the user request against the service description. The same request is compared against the service description many times and in each comparison additional conditions are added to the request until the full request is compared against the service description. The previous comparisons are abstractions of the subsequent comparisons. The main drawback of this kind of approach is that it may result in repeated re-evaluation of earlier parts of the user request, each time a successive approximation is evaluated.

For instance, suppose a mobile user is searching for a cafe with WiFi Internet, which is specified as a request definition $\exists$ sells.(Coffee $\sqcup$ (Internet $\sqcap$ $\exists$ hasComm.WiFi)). Suppose the priority of these requirements to the user, is for Internet access followed by coffee then WiFi access. Under current approximate reasoning approaches, this would result in three inference checks to a reasoner:

1. $\exists$ sells.Internet
2. $\exists$ sells.Coffee
3. $\exists$ sells.(Internet $\sqcap$ $\exists$ hasComm.WiFi)

[6]http://kaon2.semanticweb.org/ (accessed May 2009)

As such the Internet requirement is checked twice, because in step three the service description must match both Internet and ∃ hasComm.WiFi. Thus, step 3 is a continuation of step 1. However, the reasoner state at step 1 is discarded at step 2. In larger matching tasks which may be completed in realistic scenarios, this may result re-evaluation of many expressions. To alleviate the need for re-evaluation, we propose an adaptive inference strategy which incorporates modifications to the Tableaux algorithm itself. This is a different approach from current approximate reasoning techniques which perform incremental reasoning at the query level. Thus, the novel aspect of our proposed strategy, is that we modify the Tableaux algorithm to support priority driven expansion, rather than depth-first expansion, while adhering to the Tableaux transformation rules defined in Section 3.4.2. Under our proposed strategy the inference checking process be interrupted at any stage to provide a weighted degree of match result to the user, depending on user constraints such as resources or time. We will present our proposed strategy in the remainder of this chapter as a key contribution of this dissertation.

5.3 Adaptive Inference Strategy

In Section 3.5 we provided an overview of expansion for the standard Tableaux algorithm and an example. In this section we will provide an overview of the modified Tableaux algorithm functioning with our adaptive inference strategy. This section contains our modifications to enable "anytime" priority based matching. We will follow this with an example and brief complexity comparison between our adaptive strategy and the standard Tableaux algorithm.

5.3.1 Adaptive Tableaux Expansion

As discussed previously, a user request is represented as a class concept C and a service description is represented as an individual x. We use the Tableaux algorithm to prove or disprove that the inference $C(x)$ holds. The request will likely be a conjunction of many class concept definitions representing conditions, which may each contain additional conjunctions representing sub-conditions of the request. As such, a request C containing many conditions is represented as $C \equiv C_1 \sqcap C_2 \sqcap ... \sqcap C_n$, $1 \leq i \leq n$. We will assign weights representing the level of importance of each conjunct element C_i in the request later in this chapter. As outlined in Stuckenschmidt and Harmelen (2002), each conjunct element C_i subsumes the conjunction C as a whole, such that $C_1 \sqcap C_2 \subseteq C_1$ and $C_1 \sqcap C_2 \subseteq C_2$. Therefore, an inference check $(C_1 \sqcap C_2)(x)$ is equivalent to checking that $C_1(x)$ and $C_2(x)$ are each valid. A subset can be defined as $C_i \subseteq C_j$, given some class concept C_i and C_j, iff $X \subseteq Y$, where X is a set containing individuals x_r where $C_i(x_r)$ holds and Y is a set containing those individuals x_p where $C_j(x_p)$ holds.

Tableaux performs inference checking by refutation, where an inference $C(x)$ is checked by adding $\{\neg C\}$ as a type for x. A negated conjunction becomes a disjunction, such that if $C \equiv C_1 \sqcap C_2$ then $\neg C \equiv \neg C_1 \sqcup \neg C_2$. As discussed in Section 3.5 the $\sqcup$-rule which is applied to disjunctive assertions, gives rise to Tableaux tree $\mathcal{G}$ expansion. Each disjunct element of the disjunction represents an alternative expansion possibility to check. An inference is only proven if every tree expansion which depends on the request C in the inference check, generates a clash. Therefore, each conjunction in C for the inference check $C(x)$, will after negation, result in the creation of a new branch point node in $\mathcal{G}$ and each conjunct element that is a member of any of these conjunctions will result in a new branch edge in $\mathcal{G}$. Since we are associating weights of importance with each conjunct element in the request, and matching must be in conjunct priority order, this means tree expansion order will

be driven by weights rather than depth first without any specific order as in current reasoners.

Figure 5.1 illustrates the main process of our adaptive inference strategy. The boxes highlighted in yellow indicate those processes which are introduced by our proposed strategy while un-shaded boxes indicate those which are standard Tableaux operations.

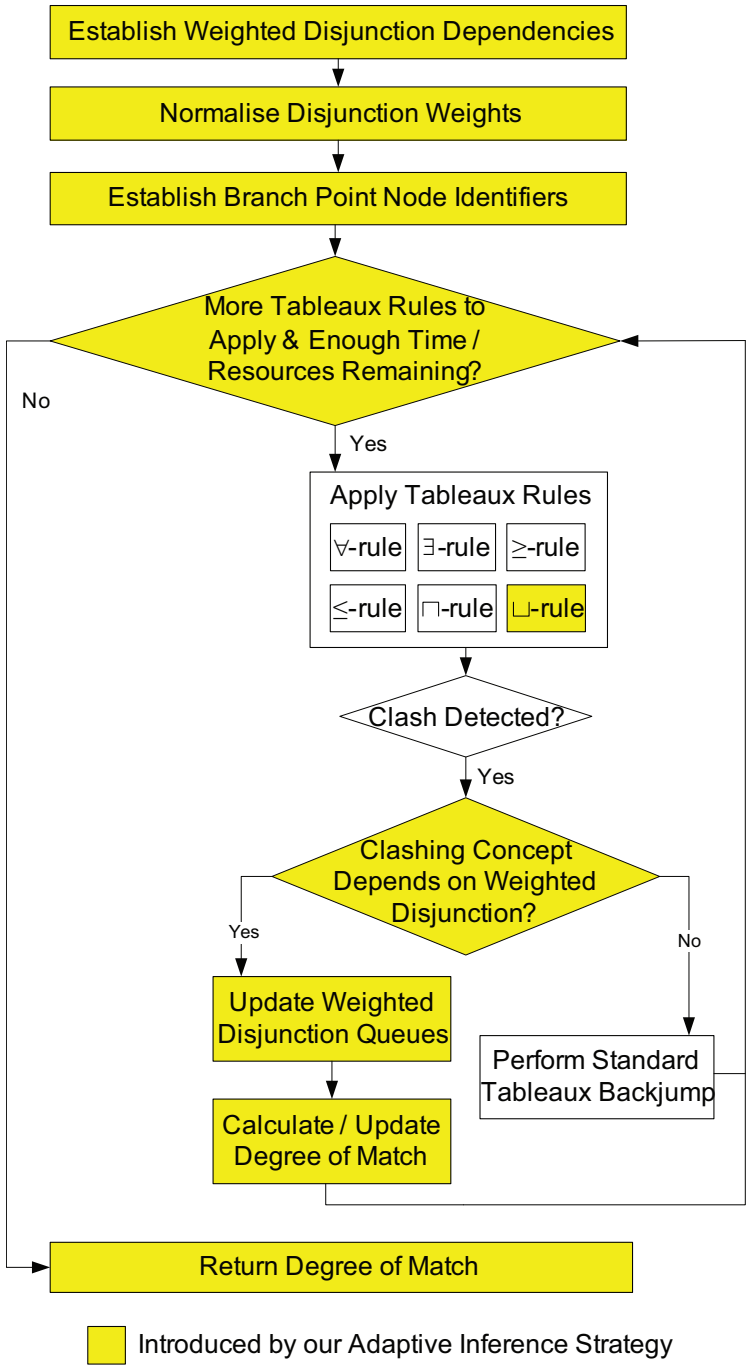


Figure 5.1: Adaptive Inference Strategy Process

The first operation in the figure establishes the dependencies between disjunctions which have been given weight / priority values by the user. These dependencies will be used by the other operations in our strategy. In the second operation the weight values associated with disjunctions are normalised so that they can be used to calculate a degree of match value. In the third operation branch point nodes, which are generated as part of Tableaux expansion, are given identifiers. Our proposed adaptive inference strategy introduces new identifiers for branch nodes. After these operations the inference reasoning process begins and continues until there are either no more Tableaux transformation rules to apply or there is insufficient time / resources to continue matching. When matching ends a degree of match value is provided. During matching the standard Tableaux transformation rules are applied (these rules were outlined in Section 3.4.2). The disjunction rule has been modified as part of our adaptive inference strategy to take account of the user specified weights, associated with disjunctions, in deciding which disjunction to apply next. This involves the use of several queues / ordering mechanisms and selection priorities. If the application of Tableaux rule generates a clash then our adaptive backjump process is invoked. If the clashing concept depends on a weighted disjunction then certain updates to the disjunction queues occur. Additionally, a clash for a weighted disjunction results in the degree of match being calculated / updated. Alternatively, some disjunctions in the ontology may not have weight values associated with them. If a clash occurs for a non-weighted disjunction then standard Tableaux backjumping occurs.

In the remainder of this section we will first formalise Figure 5.1 in algorithmic form. The remainder of the chapter will then focus on presenting each of these operations in detail. However, some of these operations will be presented in a different order than that which is shown in the figures because they rely on terminologies and assumptions described in other operations.

Algorithm 5.1 presents in algorithmic form, the main Tableaux expansion tree traversal procedure when using our proposed adaptive inference strategy, which was illustrated in Figure 5.1. This algorithm is a modification of the standard Tableaux tree traversal procedure (Algorithm 3.1) which was given in Section 3.5.1. The main difference is that Algorithm 5.1:

- calls several initialisation procedures;
- returns a degree of match for the inference check rather than a *true* or $false$ result;
- continues matching until certain constraints are not met such as available time or resources;
- utilises a modified backjumping algorithm.

Algorithm 5.1 AdapTableauxTreeTraverse($\mathcal{A}$, $\mathcal{G}$, $\neg C$, x)

Inputs: ABox $\mathcal{A}$, CompletionGraph $\mathcal{G}$, ClassConcept $\neg C$, Individual x, where $\neg C$ is the negated user request being matched to the service description x

Outputs: Decimal $degMatch$

1: $InitWeightedDisjs(\neg C, \textbf{null})$*Alg. 5.3, Section 5.4*\
2: $NormaliseWeights(\neg C)$ *Alg. 5.8, Section 5.5*\
3: $EstablishAdapBranchIDs(\neg C)$ *Alg. 5.15, Section 5.7*\
4: **while** $moreTableauxRulesToApply(\mathcal{A}, \mathcal{G}) =$ **true** and $HaltCurrInference() =$ **false** **do**
5: $clash \leftarrow ApplyTableauxRules(\mathcal{A}, \mathcal{G})$ *apply standard Tableaux rules*\
6: **if** $clash \neq$ **null then**
7: $AdaptBackJump(clash, \mathcal{A}, \mathcal{G})$
8: **end if**
9: **end while**
10: **return** $degMatch(\neg C)$ *see Section 5.8*\

The $InitWeightedDisjs$ procedure initialises the weighted disjunctions from the request and dependencies between these and will be described in Algorithm 5.3 in Section 5.4. For instance, a disjunction may contain several child disjunctions. The $NormaliseWeights$ procedure normalises the weight values associated with weighted disjunct elements and will be described in Algorithm 5.8 in Section 5.5. The $EstablishAdapBranchIDs$ procedure creates

identifiers for disjunctions which will be used as branch point identifiers. This procedure will be discussed in Algorithm 5.15 in Section 5.7.

After initialisation, Algorithm 5.1 loops while the $moreTableauxRules$-$ToApply$ (not shown) and $HaltCurrInference$ (not shown) procedures return $true$. In standard Tableaux, which we described in Section 3.5, the $more$-$TableauxRulesToApply$ function returns $false$ if no more transformation rules can be applied to the *current* branch, otherwise it returns $true$. In standard Tableaux, alternative branches are not explored unless a clash is generated for the current branch. This implies that $moreTableauxRulesToApply$ returns $false$ if a particular condition in the user request fails the matching process stops. However, our adaptive inference strategy supports partial matching, in which matching continues even if a particular condition fails. Therefore, the $moreTableauxRulesToApply$ function returns $false$ if no more transformation rules can be applied to *any* branch, otherwise it returns $true$. Thus, alternative branches are explored even if the current branch failed to generate a clash. For instance, assume a mobile user wants to find a cafe with WiFi Internet, giving rise to the negated request $(\neg\mathsf{Internet} \sqcup \neg\mathsf{WiFi}) \sqcup \neg\mathsf{Coffee}$. $moreTableaux$-$RulesToApply$ will only result $false$ if all three disjunct elements Internet, WiFi and Coffee have been expanded and applied. In Section 5.6.2 we will describe queues which maintain disjunct elements which have not yet been applied. If these queues are empty, then $moreTableauxRulesToApply$ will return $false$. In addition, as we have described in Section 5.1, our adaptive inference strategy supports "anytime" reasoning, based on user and resource constraints. Let $HaltCurrInference$ denote a function which returns $true$ if any user specified constraint is not met. For instance, if the user requires a result within 30 seconds, $HaltCurrInference$ will return $false$ if the timeout period of 30 seconds is reached / exceeded.

The *ApplyTableauxRules*[7] algorithm, applies all the transformation rules which can be applied to any assertion in the ABox $\mathcal{A}$, for the currently active branch in $\mathcal{G}$. If the mTableaux strategies, described Chapter 4, are enabled, then the transformation rules will reflect those specified in the previous chapter, otherwise they are the standard Tableaux transformation rules, with the exception of $\sqcup$-rule. The $\sqcup$-rule is modified to support our adaptive inference strategy, and will be described in Section 5.6.1 (see Algorithm 5.9). When reasoning stops either because there are no more transformation rules to apply or the process is interrupted due to user constraints, then a degree of match result is returned. Let $degMatch(\neg C)$ denote the degree of match value for the user request $\neg C$, based on the processing completed, which will be outlined in Section 5.8.

If the application of a transformation rule results in the generation of a clash, then the concept that generated the clash is returned by *ApplyTableauxRules*. If a clash has occurred the clashing concept is passed to *AdapBackJump*, which is the modified backjump algorithm, defined in Algorithm 5.2. This is the algorithmic representation of the modified backjump process which was illustrated in Figure 5.1 earlier in this section.

Algorithm 5.2 AdapBackJump($clash$, $\mathcal{A}$, $\mathcal{G}$)

Inputs: ClassConcept $clash$, ABox $\mathcal{A}$, ExpansionTree $\mathcal{G}$

1: **if** $depOnBranch(clash) =$ **null** and $depOnAdapBranch(clash) \neq$ **null** **then**
2: *depends on a branch node created by a weighted disjunction D*\
3: Let $D(x) \leftarrow createdByAssertion\Big(depOnAdapBranch(clash)\Big)$
4: $clashingConcept(D) \leftarrow$ **true**
5: $UpdateQueuesOnClash(D)$ *Alg. 5.11, Sec. 5.6.2*\
6: $UpdateResultsOnClash\Big(D, nrw(D)\Big)$ *alg 5.21, sec 5.8*\
7: **else**
8: $BackJump(clash, \mathcal{A}, \mathcal{G})$ *Alg. 3.3, Sec. 3.5.1*\
9: **end if**

In standard Tableaux, expansion occurs in depth-first order. This means when a clash occurs and the reasoner jumps back to an earlier branch and all

[7]The *ApplyTableauxRules* function applies the standard Tableaux transformation rules as outlined in Section 3.4.2 and is therefore not shown

branches and assertions which occurred after this branch can be discarded, because the branch is fully expanded. However, in our adaptive inference strategy, branch expansion is driven by weight ordering rather than depth-first. Therefore, there may still be several unfinished branches when a backjump occurs, which the reasoner will return to finish later, if there is enough time / resources. Thus, branches cannot be discarded when a backjump occurs, unless the branch is fully expanded. As a result, determining which disjunction to apply next, and therefore, which branch to expand, is handled by the $AdaptApplyDisj$ algorithm, in Section 5.6.1. If the class concept which generated a clash, depends on a weighted disjunction, no backjumping is required, because jumping to another branch is handled by $AdaptApplyDisj$. Note, the attribute $depOnBranch(C)$ denotes a standard Tableaux branch node (i.e. a branch node created as a result of the application of a non-weighted disjunction), which the assertion of a class concept C, depends on (see Algorithm 3.2 in Section 3.5.1). The attribute $depOnBranch(C)$ is $null$ if C does not depend on a standard branch node (i.e. it depends on an adaptive branch node). The attribute $depOnAdapBranch(C)$ denotes an adaptive branch node (i.e. a branch node created as a result of the application of a weighted disjunction, in our adaptive strategy) which the assertion of a class concept C, depends on. Every action, such as an assertion, is index (see Algorithm 5.9 in Section 5.6.1) and this dependency is carried forward by transformation rules such the $\forall$-rule in Algorithm 5.4 as described in 5.4.

Instead of performing standard Tableaux restore, our adaptive inference strategy calculates / updates the degree of match value for the current service being matched against the user request. This occurs using the a call to $UpdateResultsOnClash$ which is defined in Algorithm 5.21 in Section 5.8. Let $nrw(C)$ denote the relative normalised weight for the weighted disjunct C, which will be described in Section 5.5. In addition, let $clashingConcept(C)$ denote whether a particular disjunct element C of a weighted disjunction C

has generated a clash or not. This is set to *true*, when C has lead to a clash, and will be used for updating the degree of match in $UpdateResultsOnClash$. Furthermore, as we have mentioned earlier in this section, queues are used to maintain weighted disjunctions, which will be described in Section 5.6.2. A call to $UpdateQueuesOnClash$ (see Algorithm 5.11 in Section 5.6.2) is required to notify that a particular weighted disjunction element has generated a clash. This call is required because when determining which weighted disjunction to apply next, priority is given to those which have already generated clashes for some disjunct elements.

However, in addition to weighted disjunctions, which are derived from the user request, there may also be other disjunctions which can be evaluated by the $\sqcup$-rule. These disjunctions may exist somewhere else in the ontology and are only applied after no more weighted disjunctions can be applied to a particular branch expansion. In the case that a clash is generated by a class concept, which directly depends on one of these non-weighted disjunctions, then the standard Tableaux backtracking is used (see Algorithm 3.3 in Section 3.3). This continues until all expansions for a non-standard disjunction generate a clash, thereby generating a clash for any weighted disjunction which this depends on.

In this section we have provided an overview of the Tableaux algorithm which has been modified to reflect our adaptive inference strategy, which performs expansions in weight order. In the next section we will illustrate the main operation of our adaptive inference strategy using an example, before outlining the details of our strategy in more detail.

5.3.2 Adaptive Tableaux Expansion Example

In this section, we will provide an example of an inference check using our adaptive inference strategy, which was described in the previous section. This example can be contrasted with the standard depth-first Tableaux example we

provided in Section 3.5.2. In this example, we will perform the inference check on the same ABox was used in Section 3.5.2 and provided again in Figure 5.2. We will also perform the same inference check on this ABox, except that we will associate weights with each of the conditions in the request. The example checks whether $C(x)$ holds, where $\neg C \equiv ((\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3)$ and the associated weights are listed in as listed in Table 5.1.

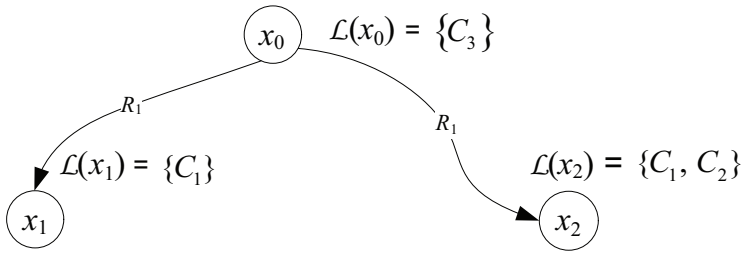


Figure 5.2: Adaptive Inference Example ABox

Class Concept Definition	Explicit Weight w	Relative Weight rw	Normalised Relative Weight nrw
$\neg\forall R_1.(\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3$	1.0	1.0	
$\neg\forall R_1.(\neg C_1 \sqcup \neg C_2)$	1.0	1.0	
$\neg C_1$	1.0	1.0	0.5
$\neg C_2$	0.4	0.4	0.2
$\neg C_3$	0.6	0.6	0.3
Total			1.0

Table 5.1: Example Adaptive Inference Weight Values

The explicit weight is the weight given to each requirement in the request by the user, while relative and normalised weights are generated by our adaptive inference strategy. Weight establishment will be discussed in Section 5.5. For this section, it is sufficient to understand that explicit weight values w are allocated to the user to specified conditions in the request. Relative rw and normalised nrw weights are calculated from explicit weights w by the adaptive inference strategy. Relative weight rw values are used to determine the order

in which disjunct elements are applied and normalised relative weight nrw values are used to determine the degree of match. A nrw is given only to disjunct elements which do not expand into additional sub-disjunctions and the sum of all nrw values in the request must always equal 1. Thus, the tree is expanded in rw order. However, in addition to this, the $\forall$-rule may copy a disjunction and add it as a type to several different individuals, and only one disjunction must generate a clash for all disjunct elements to prove the condition. Therefore, if disjunct elements of a disjunction have been applied, and have generated a clash, then obviously the next disjunct element of the same disjunction should be applied in preference to the first disjunct element of a copy of the same disjunction.

In Section 5.6.2 we will establish three different queues to maintain these priorities. For the purposes of the section, we will provide informal explanation to understand the example. Let $\mathcal{CQ}_u$ denote a queue containing disjunctions which have not yet been applied. Let $\mathcal{CQ}_{pn}$ denote a queue containing disjunctions where at least one disjunct element has been applied, but has not yet generated a clash. Let $\mathcal{CQ}_{pa}$ denote a queue containing disjunctions where at least one disjunct element has been applied and all disjunct elements which have been applied have also generated a clash. $\mathcal{CQ}_u$ is ordered in descending weight order and $\mathcal{CQ}_{pa}$ in ascending weight order, where the weight for a disjunction represents the relative rw of the next applicable disjunct element for the disjunction. The $\mathcal{CQ}_{pa}$ queue is favoured over $\mathcal{CQ}_u$ if the next applicable disjunction has a weight which is lower than the next disjunction in $\mathcal{CQ}_u$. Table 5.2 illustrates the contents of the queues at the beginning of each expansion step, and the chosen disjunction to apply at each step is highlighted in yellow.

Each expansion step is shown in Figure 5.3. In the figure, shaded nodes represent nodes which were already created by a previous step, while non-shaded nodes indicate those created in the current the step. We use different branch node identifiers for our adaptive inference strategy. Branch node identifiers

Step	Queue	Disjunction	Next Element	Next rw
a	$\mathcal{CQ}_u$	$(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3 \in \mathcal{L}(x_0)$	$\forall R_1.(...)$	1.0
b	$\mathcal{CQ}_u$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$	$\neg C_1$	1.0
		$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$	$\neg C_1$	1.0
	$\mathcal{CQ}_{pn}$	$(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3 \in \mathcal{L}(x_0)$	$\neg C_3$	0.6
c	$\mathcal{CQ}_{pa}$	$(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3 \in \mathcal{L}(x_0)$	$\neg C_2$	0.4
		$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$	$\neg C_1$	1.0
	$\mathcal{CQ}_u$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$	$\neg C_1$	1.0
d	$\mathcal{CQ}_{pa}$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$	$\neg C_2$	0.4
	$\mathcal{CQ}_u$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$	$\neg C_1$	1.0
e	$\mathcal{CQ}_u$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$	$\neg C_1$	1.0
f	$\mathcal{CQ}_u$	$\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$	$\neg C_2$	0.4

Table 5.2: Example Adaptive Inference Expansion Order

appear in subscript in the figure which are determined based on the depend and breadth of the branch node. Additionally, if the $\forall$-rule (see Section 3.4.2) generates multiple equivalent copies of a disjunction, the branch nodes created from these disjunctions are given identifiers with a third copy value. Branch identifiers for our adaptive inference strategy will be explained in detail in Section 5.7.

As shown in Figure 5.3 step a, the $\sqcup$-rule is first applied to the disjunction $(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3$ added as a type to x_0 because it has a weight of 1.0. From this disjunction, the particular disjunct element which is added as a type to x_0, is $\forall R_1.(\neg C_1 \sqcup \neg C_2)$ because this element has a weight of 1.0, which is higher than the $\neg C_3$ which is 0.6. Application of this element results in the creation of the branch node b_{3-0} which has an identifier with a depth value that has been incremented by one when compared to the branch node b_{2-0} to which it depends. After the application of the disjunct element a clash has not occurred. Therefore, $(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3$ is stored in $\mathcal{CQ}_{pn}$ which contains disjunctions where not all applied disjunct elements have clashed. Application

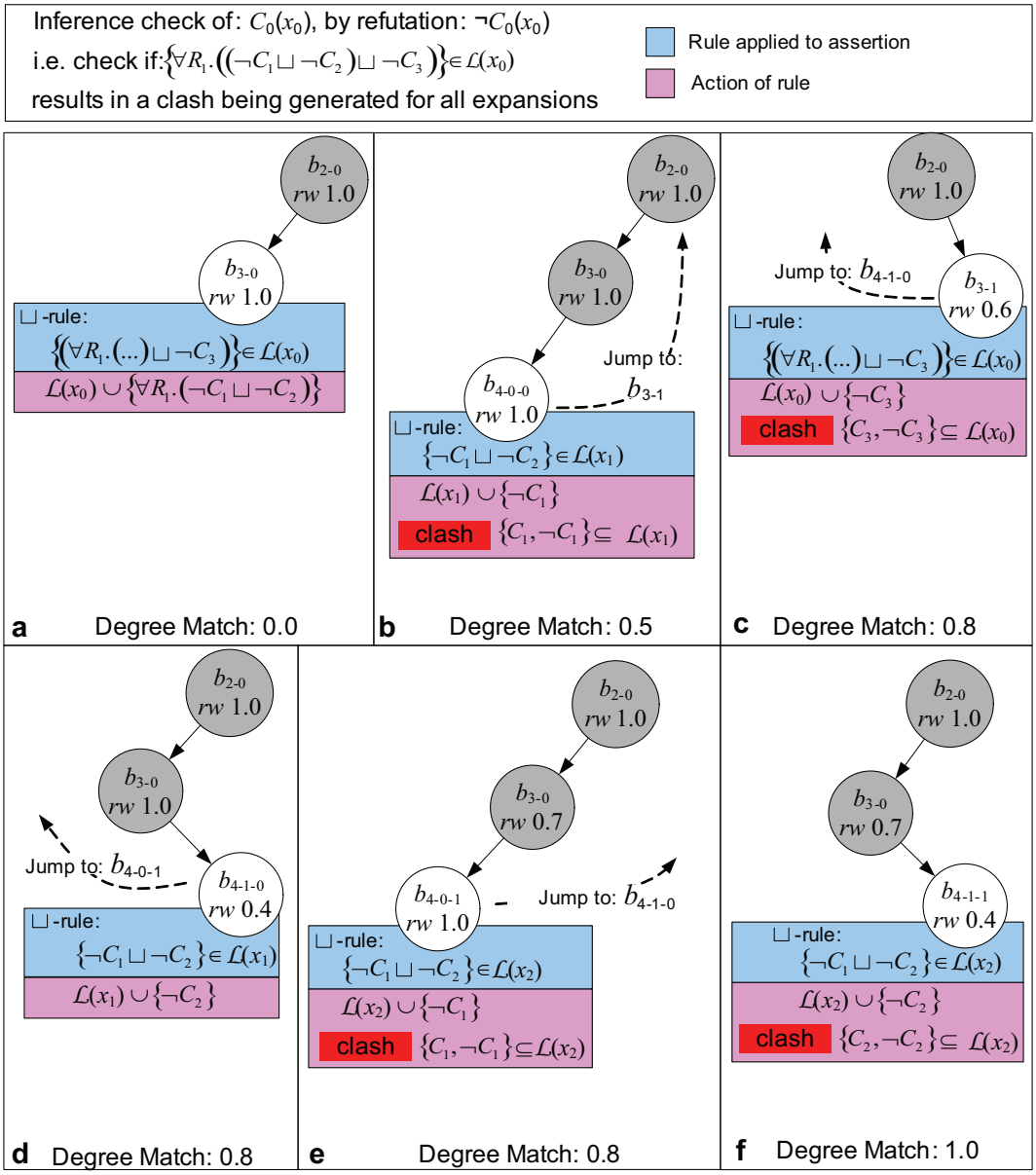


Figure 5.3: Adaptive Inference Expansion Search Tree

of the $\forall$-rule will add $\neg C_1 \sqcup \neg C_2$ as a type to the individuals x_1 and x_2 (not shown).

In step b, the disjunction $\neg C_1 \sqcup \neg C_2$, added to x_1, is applied. The disjunct $\neg C_1$ is applied first and added as a type to x_1 because it has a weight of 1.0. This results in the creation of a branch node b_{4-0-0} which has an incremented depth value compared to b_{3-0}. In addition, the identifier of b_{4-0-0} contains a third index value of zero used to differentiate between the two

equivalent copies of $\neg C_1 \sqcup \neg C_2$ which were generated by the $\forall$-rule. Application of the disjunct element at step b generated a clash, which results in the disjunction $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$ and $(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3 \in \mathcal{L}(x_0)$ being moved to the queue $\mathcal{CQ}_{pa}$ because this queue contains disjunctions where all applied disjunct element members have generated a clash. In addition, since the clashing concept $\neg C_1$ depends on a weighted disjunction, a standard Tableaux backtrack does not occur. Rather, the adaptive inference strategy selects the next disjunction to be applied. The next applicable disjunct is $\neg C_3$ from $(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3$, which depends on b_{2-0}. Therefore, the reasoner will change back to branch b_{2-0} without discarding any branch nodes from step b because these branch nodes are unfinished (i.e. there are more nodes to apply which depend on b_{3-0}).

In step c, $(\forall R_1.(\neg C_1 \sqcup \neg C_2)) \sqcup \neg C_3 \in \mathcal{L}(x_0)$ is applied because the next element has the lowest weight in $\mathcal{CQ}_{pa}$ (implying fewer disjunct elements remaining to prove the inference). $\neg C_3$ is added to x_0 which generates a clash, and the disjunction is removed from the queues because all elements have now been applied.

In step d the next applicable disjunction is $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$ because the next element $\neg C_2$ has a weight of 0.4 which is lower than the next element for the disjunction in $\mathcal{CQ}_u$. Since $\neg C_2$ is the second element, the new branch node identifier b_{4-1-0} has an incremented breadth value compared to b_{4-0-0} from step b. When this next disjunction is applied the element $\neg C_2$ is added as a type to x_1 and the disjunction is removed from the queues because all elements have now been applied. However, this did not generate a clash.

In step e the other copy of $\neg C_1 \sqcup \neg C_2$ (i.e. the copy added to x_2), is applied. This results in the creation of a branch node b_{4-0-1} which has an incremented copy value compared to b_{4-0-0} from step b. Both $\neg C_1$ and $\neg C_2$ generate of this disjunction generate a clash for x_2, as shown in steps e and f.

Note that both $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$ and $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$, depend on the same disjunct element $\forall R_1.(\neg C_1 \sqcup \neg C_2) \in \mathcal{L}(x_0)$, so if either disjunction generates a clash for all expansions then $\forall R_1.(\neg C_1 \sqcup \neg C_2)$ is proven. If $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_1)$ had generated a clash for both elements as shown in steps b and d, then steps e and f would not have been required. However, since step d did not generate a clash, $\neg C_1 \sqcup \neg C_2 \in \mathcal{L}(x_2)$ had to be evaluated in steps e and f.

The inference check generated a clash for every expansion, therefore, the final degree of match returned to the user is 1.0, which indicates a full match. This is obtained by adding the normalised relative weight nrw, of every disjunct element listed in Table 5.1, which generated a clash (i.e. 0.5 + 0.3 + 0.2 = 1.0). Note, element $\neg C_1$, which has a nrw of 0.5, was only added to the degree of match once, even though it generated a clash for two different individuals. This is because only one equivalent element of $\neg C_1$ can be used in the degree of match, because they belong to disjunctions which both depend on the same definition (i.e. the universal quantifier). One of the key benefits of the adaptive reasoning strategy is that the reasoning could have been stopped at any one of the steps. For instance, the degree of match at step c, was 0.8, because after step c, both C_1 and C_3 were found to clash, and the sum of their nrw is 0.8. It took until step f, to find an expansion where both C_1 and C_2 generated a clash, giving the degree of match result of 1.0.

Based on this example, we hypothesise that a good indication of the degree of match for a service may be obtained very early in the matching process. If due to time or resource constraints it is not possible to complete the expansion process required to find an expansion which produced a full match, then a partial degree of match result is much more informative to the user than no result. In addition, since the most important requirements are matched first (based on user assigned priorities / weights) the degree of match result of 0.8 was found at step c. This is contrary to the example in Section 3.5.2 where standard depth-first Tableaux expansion was used, where C_3 was checked in the

last step, even though it was more important than C_2. Therefore, at step c, the degree of match would have been only 0.5 using standard Tableaux, compared to 0.8 using our priority based depth first approach. In the remaining sections of this chapter, we provide the detail for the functionality of our adaptive inference strategy, which was illustrated in this section.

Now that we have provided an overview the modifications to the standard Tableaux algorithm to enable adaptive inference and an example of its operation, in the next section we provide a brief complexity comparison between our proposed adaptive inference strategy and standard Tableaux.

5.3.3 Algorithm Complexity Comparison

In Section 3.5.1 we outlined the tree expansion process of standard Tableaux. Standard Tableaux is a depth-first expansion procedure, which has a worst case complexity of $O(n)$ where n is the total number of branch point nodes which can be expanded in the expansion tree $\mathcal{G}$. One of the goals of our proposed adaptive inference strategy is that matching can be stopped prematurely, depending on user constraints such as available resources or time. However, if the matching task is allowed to complete in full without exceeding these constraints then it will have the worst case complexity of $O(n)$ which is the same as standard Tableaux. Standard Tableaux has a best case complexity of $O(m)$ where m is the average number of nodes in a branch in the expansion tree $\mathcal{G}$. For instance, when matching a user request against a service description which does not match any requirement in the user request, standard Tableaux will fully expand one branch expansion of the tree until no more transformation rules can be applied, then report a failed match. It will not continue searching for alternative branches, because the first expansion failed to generate a clash. Alternatively, our adaptive inference strategy, may be stopped early due to user constraints such as time or resources. Therefore, our proposed strategy

has a best case complexity of $O(1)$. For instance, matching may stop after only one expansion.

Moreover, the main idea behind our adaptive inference strategy is to change the order in which expansions occur, and allow matching to be stopped prematurely. If all conditions in the user request are checked and the matching task finishes to completion it is likely that the complexity will be the same as standard Tableaux. However, since mobile users are operating in extremely dynamic and constrained environments where resources and time are extremely limited, these constraints must be considered as important inputs to the matching process.

In the remaining sections of this chapter we will describe in more detail the way in which weighted disjunctions are represented, the mechanisms for deciding which disjunction to select, management of reasoner state and degree of match calculations. In the next section we will describe how weighted disjunctions are represented in our adaptive inference strategy.

5.4 Weighted Disjunction Dependencies

In our adaptive strategy, we distinguish between weighted disjunctions and non-weighted / standard disjunctions. Weighted disjunctions are those which are derived from the user request and are associated with a weight. Any other disjunctions, which might otherwise exist in the knowledge base are considered to be non-weighted disjunctions and are applied only after no more weighted disjunctions can be applied in a particular expansion. In this section we define the notation which we will use when referring to weighted disjunctions throughout this chapter. Non-weighted disjunctions are applied in the standard way as described previously in Section 3.5. In Description Logic (see Section 3.3) a standard disjunction is of the form $D \equiv E_1 \sqcup (E_2 \sqcap E_3)$, where D is a disjunction and E_1 and $(E_2 \sqcap E_3)$ are considered to be disjunct elements of the disjunction D. In addition, assume $E_2 \equiv E_4 \sqcup E_5$, $E_3 \equiv E_6 \sqcup E_7$. This

implies that the disjunct element $(E_2 \sqcap E_3)$ of D contains additional conjuncted disjunctions $E_4 \sqcup E_5$ and $E_6 \sqcup E_7$. In this section we will establish object and property notation for weighted disjunctions for ease of description in the remainder of this chapter. We will also re-iterate the situations in which a clash is generated for a disjunct element which has multiple sub-disjunctions which depend on it later in the section.

In our adaptive inference strategy, if D is a weighted disjunction, implying it was derived from the user request and has a weight associated with it, then D can be considered to be an instance of an object $WeightedDisjunction$ and the disjunct elements E_1, $(E_2 \sqcap E_3)$, E_4, E_5, E_6 and E_7 can all be considered to be instances of an object $WeightedDisjunct$. These objects and their dependencies which we will define last in this section, are illustrated in Figure 5.4.

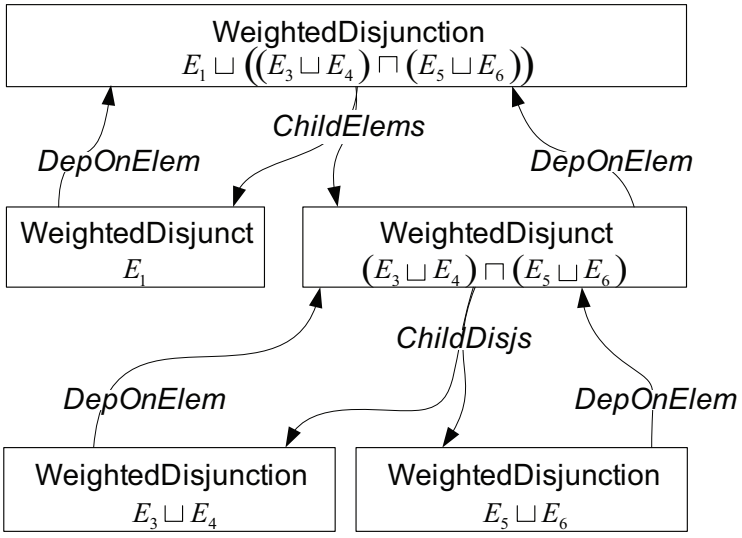


Figure 5.4: Weighted Disjunction Dependencies

Let $D.ChildElems$ denote a property containing these disjunct element members of a disjunction D. For instance, in the example above, the disjunction D has child elements, such that $D.ChildElems = \{E_1, (E_2 \sqcap E_3)\}$. In addition, a disjunct element member E can itself contain a disjunction, or several

disjunctions as a conjunction. Let $E.ChildDisjs$ denote the property containing the set of child disjunctions for an element E. For instance, in the above example the disjunction D, contains a disjunct element member $(E_2 \sqcap E_3)$ which is a conjunction and both E_2 and E_3 are disjunctions, where $E_2 \equiv (E_4 \sqcup E_5)$ and $E_3 \equiv (E_6 \sqcup E_7)$. Therefore, $(E_2 \sqcup E_3).ChildDisjs = \{(E_4 \sqcup E_5), (E_6 \sqcup E_7)\}$. We also establish a property for the reverse dependency, in order to identify the disjunction which a disjunct element is a member of, and thus depends on. Let $E.DepOnDisj$ denote a property the disjunction which a disjunct element E depends on. For instance, in the example above, the disjunct element $(E_2 \sqcap E_3)$ is a member of the disjunction D, therefore, $(E_2 \sqcap E_3).DepOnDisj = D$. Furthermore, since a disjunct element can contain, one or more sub-disjunctions let $W.DepOnElem$ denote the reverse property, such that $W.DepOnElem$ denotes the disjunct element which a sub-disjunction W depends on. For instance, in the example above the disjunction $E_4 \sqcup E_5$ and $E_4 \sqcup E_5$ both depend on the disjunct element $(E_2 \sqcap E_3)$, therefore, $(E_4 \sqcup E_5).DepOnElem = (E_2 \sqcap E_3)$ and $(E_6 \sqcup E_7).DepOnElem = (E_2 \sqcap E_3)$.

In our strategy, a $WeightedDisjunct$ instance is considered to be equivalent to / contain the class concept definition which it represents. For instance, suppose there is a user request for an Internet cafe selling WiFi Internet or Internet on provided desktop PCs, defined as $\mathsf{Request} \equiv \exists\ \mathsf{sells}.\ \mathsf{InternetRequest}$, where $\mathsf{InternetRequest} \equiv ((\mathsf{WiFi} \sqcap \mathsf{Internet}) \sqcup (\mathsf{Desktop} \sqcap \mathsf{Internet}))$. Since Tableaux proves inferences by negation, the negated request is $\neg\mathsf{Request} \equiv \forall\ \mathsf{sells}.\ \neg\mathsf{InternetRequest}$, where $\neg\mathsf{InternetRequest} \equiv ((\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet}) \sqcap (\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet}))$. In our proposed adaptive inference strategy, this request is a $WeightedDisjunct$ instance equal to the definition $\forall\ \mathsf{sells}.\neg\mathsf{InternetRequest}$. This instance has two child $WeightedDisjunction$ instances that can be expanded from it, which are $\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet}$ and $\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet}$. However, although there is a dependency from the $WeightedDisjunct$ to two child $WeightedDisjunction$ instances, due to the existence of definitions other than

disjunctions, it is possible that one or more transformation rules (i.e. other than the $\sqcup$-rule) are applied after the $WeightedDisjunct$ has been evaluated but before the child disjunctions are evaluated. For instance, the $\forall$-rule is applicable to the definition $\neg\mathsf{Request} \equiv \forall\mathsf{sells}.\ \neg\mathsf{InternetRequest}$, which expands $\neg\mathsf{InternetRequest}$. Following this, $\neg\mathsf{InternetRequest} \equiv ((\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet}) \sqcap (\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet}))$ is a conjunction, which the $\sqcap$-rule expands into two separate weighted disjunctions $(\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet})$ and $(\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet})$. Only now is the $\sqcup$-rule applicable to each of the weighted disjunctions. For the purposes of the algorithms which we describe in this section, we need a mechanism to denote the last weighted disjunct element which a class concept definition depends on (if any). Let $depOnAdapDisjElem(C_i)$ denote the last weighted disjunct element which the class concept C_i depends on. For instance, in the above example, $depOnAdapDisjElem(\neg\mathsf{InternetRequest}) = \neg\mathsf{Request}$ and $depOnAdapDisjElem(\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet}) = \neg\mathsf{Request}$.

Algorithm 5.3 is the procedure for initialising the weighted disjunction dependencies which have been defined in this section. This procedure is called by the modified Tableaux adaptive inference tree traverse Algorithm 5.1 which was presented earlier in Section 5.3.1. The negated class definition which represents the user request is passed as the first parameter C_i to this algorithm and a *null* value is initially passed to the second parameter $lastElem$. This second parameter will be used to associate the most recent weighted disjunct element, with the current class concept definition C_i being evaluated by the algorithm.

Firstly, the class concept C_i is set to depend on the last weighted disjunct element $lastElem$. In the case that C_i is a disjunction, an instance of $WeightedDisjunction$ is created to represent this disjunction. In addition, a new unique identifier is created to fill $D.ID$. We will explain this $D.ID$ later in this section. D is set to depend on $lastElem$ which has been passed as a parameter to the algorithm (initially *null*). The new disjunction D is added

Algorithm 5.3 InitWeightedDisjs(C_i, $lastElem$)

Inputs: ClassConcept C_i, WeightedDisjunct $lastElem$

1: Let $depOnAdapDisjElem(C_i) \leftarrow lastElem$
2: **if** $C_i = (C_1 \sqcup ... \sqcup C_m)$ **then**
3: Let D be an instance of $WeightedDisjunction$
4: Let $D.ID$ be a new unique identifier
5: Let $D.DepOnElem \leftarrow lastElem$
6: $lastElem.ChildDisjs \leftarrow lastElem.ChildDisjs \cup \{D\}$
7: **for all** C_j where $C_i = (C_1 \sqcup ... \sqcup C_m)$, $1 \leq j \leq m$ **do**
8: Let E be an instance of $WeightedDisjunct$
9: Let $depOnAdapDisjElem(C_j) \leftarrow E$
10: $D.ChildElems \leftarrow D.ChildElems \cup \{E\}$
11: Let $E.DepOnDisj \leftarrow D$
12: Let $E.ID$ be a new unique identifier
13: $InitWeightedDisjs(C_j, E)$
14: **end for**
15: **else if** $C_i = \forall R.C_j$ or $C_i = \exists R.C_j$ **then**
16: $InitWeightedDisjs(C_j, lastElem)$
17: **else if** $C_i = (C_1 \sqcap ... \sqcap C_m)$ **then**
18: **for all** C_j where $C_i = (C_1 \sqcap ... \sqcap C_m), 1 \leq j \leq m$ **do**
19: $InitWeightedDisjs(C_j, lastElem)$
20: **end for**
21: **end if**

to the set of child disjunctions for the element which D depends on. Then the algorithm loops on all disjunct elements of C_i and creates a new $Weighted\text{-}Disjunct$ object E to represent each and adds it to the set of child elements for the new disjunction D. Each disjunct member definition of D, is associated with the new weighted disjunct E object. The new weighted disjunction E is set to depend on the disjunction D, and a new unique identifier is created and stored in $E.ID$. We will explain this $E.ID$ later in this section. Then there is a recursive call back to Algorithm 5.3 for each disjunct element, and the element as well as the new disjunction D which the element depends on, are passed as parameters. This recursion ensures that the algorithm creates the tree of dependencies for all sub-disjunctions contained in the user request. In the case that the parameter C_i is a universal or existential quantifier, such as $\forall R.C_j$, the role filler C_j of the quantifier is recursively examined by the algorithm, because application of a $\forall$-rule or $\exists$-rule will add the role filler C_j as a type to

one or more individuals, and this role filler C_j could contain a sub-disjunction. In the case that the parameter C_i is a conjunction, each conjunct element of the conjunction are recursively evaluated, because these elements could also contain sub-disjunctions.

As we described in Section 3.4.2 and 3.5.1 each disjunct element E member of a disjunction D represents an alternative expansion, and all expansions of D must generate a clash in order to prove that D itself generates a clash. Therefore, all alternatives of a disjunction need to be explored, requiring the use of the branching expansion tree $\mathcal{G}$. Conversely, as described in Section 3.4.2, only one expansion of a conjunction C_i needs to generate a clash, in order to prove that the conjunction C_i clashes. For instance, in the example we just described earlier in this section, the conjunction $\neg\mathsf{InternetRequest}$ contains two conjunct elements which can be expanded. Each of these conjunct elements are themselves disjunctions $(\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet})$ and $(\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet})$. Only one of these disjunctions needs to clash for all expansions, to prove the conjunction $\neg\mathsf{InternetRequest}$, which they depend on, generates a clash. In addition, $\neg\mathsf{InternetRequest}$ depends on the weighted disjunct $\neg\mathsf{Request}$. Therefore, if $\neg\mathsf{InternetRequest}$ generates a clash then $\neg\mathsf{Request}$ is considered to generate a clash as well. Thus, if any one of these conjunctive disjunctions generates a clash for of its all expansions, the other disjunction (in the conjunction) does not need to be evaluated. Suppose the sub-disjunction $(\neg\mathsf{Internet} \sqcup \neg\mathsf{WiFi})$, was checked first and generated a clash for both disjunct elements as shown in Figure 5.5. This proves that the conjunction clashes. The second disjunction $(\neg\mathsf{Internet} \sqcup \neg\mathsf{Desktop})$, would not need to be checked, because the conjunction is proven. In the figure, class concept definitions which have been found to clash are highlighted in yellow.

Additionally, the same disjunction may be copied and added as a type to many different individuals by the application of a $\forall$-rule to a universal quantifier class concept definition. All copies then depend on this universal quantifier,

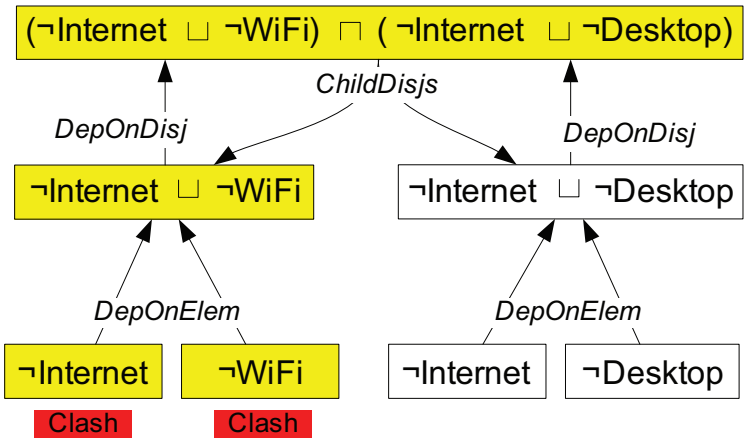


Figure 5.5: Conjunction Dependency Example

thus if any copy generates a clash for all expansions, this proves the quantifier clashes, and the others do not need to be evaluated. Therefore, we establish a unique identifier ID which is shared between all equivalent copies of a $WeightedDisjunction$ instance, which have been copied by a $\forall$-rule and depend on the same definition. Let $D.ID$ denote a property containing a unique identifier which is shared between all equivalent copies of a disjunction D. If a copy of D is added as a type to several different individuals by the $\forall$-rule, then all copies of share the same $D.ID$. In addition, we also establish a dependency between all equivalent copies of the disjunct element members. That is, since a disjunction D^j has disjunct element E_i members, such that $D^j \equiv E_1 \sqcup ... \sqcup E_n$, $1 \leq i \leq n$, then if there are m equivalent copies of the disjunction D^j, then there are also m copies of each disjunct element E_i, such that equivalent copies of the element E_i include $\{E_i^1, E_i^2 ..., E_i^m\}$, $1 \leq j \leq m$. Each copy of E_i shares the same identifier, such that $\{E_i^1.ID = E_i^2.ID$, $E_i^2.ID = E_i^3.ID$, ..., $E_i^{m-1}.ID = E_i^m.ID\}$. This implies that an E_i^1 and an E_i^2 can only share the same ID such that $E_i^1.ID = E_i^2.ID$, iff the disjunctions which they depend on, also share the same ID, such that $E_i^1.DepOnDisj.ID = E_i^2.DepOnDisj.ID$. Disjunction and disjunct equivalent copy identifiers are illustrated in Figure 5.6, where the shaded boxes indicate equivalent identifiers ID and the non-shaded boxes indicate weighted disjunctions and disjunct elements.

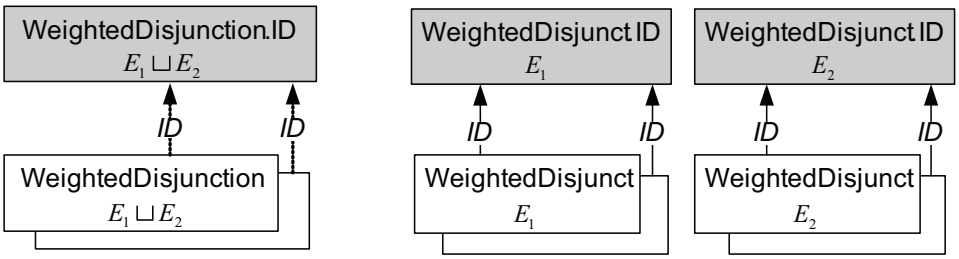


Figure 5.6: Weighted Disjunction Unique Identifier

Now we will illustrate this using an example. A mobile user searching for a WiFi Internet cafe defines the request containing $\exists$ sells.(Internet $\sqcap$ WiFi) and this request is matched against the service description individual netcafe. This request definition is negated to give $\forall$ sells.($\neg$Internet $\sqcup$ $\neg$WiFi) and added as a type for the individual netcafe. Assume that netcafe is connected the individuals inet and cakes by the role sells, and inet is a member of the classes Internet and WiFi, while cakes is not. Applying the $\forall$-rule on the definition $\forall$ sells.($\neg$Internet $\sqcup$ $\neg$WiFi) (which is a type for inet), will add the disjunction ($\neg$Internet $\sqcup$ $\neg$WiFi) as a type to both inet and cakes. inet will generate a clash for both $\neg$Internet and $\neg$WiFi because it contains their negation as a type, while cakes will not. Both copies of the disjunction ($\neg$Internet $\sqcup$ $\neg$WiFi) will share the same ID value because they are equivalent copies of the same disjunction. This is illustrated in Figure 5.7, where clashing elements are highlighted in yellow.

The $\forall$-rule is applied to a universal quantifier of the form $\forall R.C_j$. In the case that the role filler C_j can be expanded to a weighted disjunction D, this weighted disjunction D will need to be copied to potentially multiple individuals. All copies of D will share the same $D.ID$. Therefore, when a $\forall$-rule is applied, our proposed adaptive strategy will set a marker variable for the disjunct element E which a class concept C_j depends on, to indicate that child disjunctions of E will need to be copied. Algorithm 5.4 shows the procedure for the $\forall$-rule, using our adaptive strategy. The algorithm receives

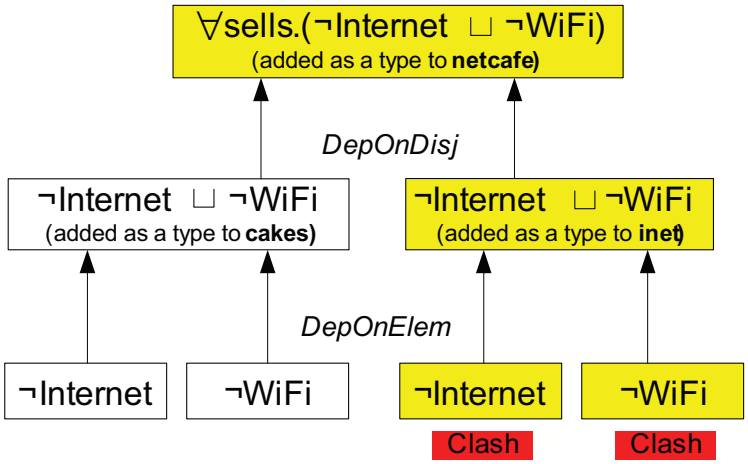


Figure 5.7: Universal Quantifier Dependency Example

as input C_i which is a universal quantifier definition, which is a type for the individual x_r, where x_r is in the ABox $\mathcal{A}$.

Algorithm 5.4 ApplyUnivRule(C_i, x_r, $\mathcal{A}$)

Inputs: ClassConcept C_i, Individual x_r, ABox $\mathcal{A}$
Outputs: ClassConcept *clash*
Pre-conditions: $C_i \equiv \forall R.C_j$

1: **for all** x_p, where $R \in \mathcal{L}(\langle x_r, x_p \rangle)$, $1 \leq p \leq n$ **do**
2: **if** $n > 1$ and $depOnAdapDisjElem(C_i) \neq$ **null then**
3: Let $lastElem \leftarrow depOnAdapDisjElem(C_i)$
4: $copyAndAssociateChildDisjs(lastElem) \leftarrow$ **true**
5: **end if**
6: $depOnAdapDisjElem(C_j) \leftarrow depOnAdapDisjElem(C_i)$
7: $AddType(C_j, x_p, \mathcal{A})$ *Add C_j as a type for x_p, where $C_i \equiv \forall R.C_j$*\
8: **end for**

The input class concept C_i is a universal quantifier of the form $\forall R.C_j$. The algorithm loops for all individuals x_p for which the role filler definition C_j will added to as a type. As stated above, if the universal quantifier C_i, where $C_i \equiv \forall R.C_j$, depends on a weighted disjunction, if C_j can be expanded to a disjunction D and if there is more than one individual x_p which C_j is being added as a type to, then a copy of the disjunction D will be made when it is encountered (by the *AddType* algorithm which is provided later in this section). Let $copyAndAssociateChildDisjs(E)$ denote a marker which can be set to *true* in order to indicate that D will need to be copied, where E is

the weighted disjunct element which contains D as a child disjunction. This algorithm also updates the weighted disjunct dependency for C_j in case this has changed (due to an earlier copy). The class concept C_j is set to depend on the same weighted disjunct E which C_i depends on, thereby, carrying this dependency from C_i to C_j. We also assume that these dependencies are carried forward in the same way for other rules such as the $\exists$-rule and the $\sqcap$-rule. For instance, if the $\sqcap$-rule is applied to a conjunction C_i, where $C_i \equiv C_1 \sqcap ... \sqcap C_n$, $1 \leq j \leq n$, then $depOnAdapDisjElem(C_j) \leftarrow C_i$, for all conjunct members C_j of the conjunction C_i.

Finally, we invoke the *AddType* function, which is defined in Algorithm 5.5. This algorithm is used by our proposed inference reasoning strategy, whenever a transformation rule adds a type C to an individual x, as was is the case in the $\forall$-rule detailed in Algorithm 5.4. The main reason for this algorithm, is that it copies any $WeightedDisjunction$ instances using the function $CopyDisjunction$ (defined in Algorithm 5.6), which need to be copied (i.e. if $copyAndAssociateChildDisjs$ is set) when these are added as a type to a particular individual, as well as adding the C as a type for x. More specifically, if the class concept being added is actually a weighted disjunction, then a copy of this disjunction is made if the $copyAndAssociateChildDisjs(E)$ has been set to *true* for the disjunct element E which C depends on, by the $\forall$-rule. Alternatively, if E is a disjunct element which is a member of a disjunction which has been copied earlier due to a $\forall$-rule, then its child disjunctions are copied and the dependencies updated. Let $copied(B)$ denote a Boolean property which is set to *true*, if a weighted disjunct element B or disjunction B has been copied previously.

A $WeightedDisjunction$ instance and all of its disjunct elements are copied by the $CopyDisjunction$ procedure, which is defined in Algorithm 5.6. The first argument of $CopyDisjunction$ is the weighted disjunction that needs to be copied, the new copy will be set to depend on the weighted disjunct element

Algorithm 5.5 AddType(C, x, $\mathcal{A}$)

Inputs: ClassConcept C, Individual x, ABox $\mathcal{A}$
Outputs: ClassConcept *clash*

1: **if** C is an instance of $WeightedDisjunction$ and
   $copyAndAssociateChildDisjs(C.DepOnElem) =$ **true then**
2: $C \leftarrow CopyDisjunction\big(C, depOnAdapDisjElem(C), \textbf{false}\big)$
3: **else if** C is an instance of $WeightedDisjunct$ and
   $isACopy(C) =$ **true then**
4: **for all** $D \in C.ChildDisjs$ **do**
5: $CopyDisjunction\big(D, C, \textbf{true}\big)$
6: **end for**
7: **end if**
8: $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{C\}$ *do standard Tableaux add type*\

passed to the second argument, and the third argument denotes whether new unique identifiers ID should be generated. If the ∀-rule has resulted in the copying of the disjunction, then all copies share the same identifiers ID because they will depend on the same universal quantifier, so the third argument is set to $false$. Alternatively, if the disjunction is being copied because a parent disjunction which it depends on was copied earlier by the ∀-rule, then a new identifier is generated for each copied disjunction (and their elements), so the third argument is $true$. They need to be copied because they depend on different disjunctions, but will not share the same identifier ID.

In this algorithm, let $copyCount(D)$ denote an index indicating the number of copies of the original disjunction D which have been made. If this has not been initialised this means that the disjunction D, is the first copy and is, therefore, set to an index of zero. If D is the first / original copy, then a copy does not need to be made. The copy count index is passed to a function $UpdateCopyNbr$, which will update the identifier for each of the disjunct element members of the disjunction, which is used for identifying branch nodes b in the expansion tree $\mathcal{G}$. We will define the $UpdateCopyNbr$ in Algorithm 5.16 in Section 5.7.1 where we discuss adaptive branch identifiers. If no copy needs to be made the disjunction D is then returned. However, if a copy does need

Algorithm 5.6 CopyDisjunction(D, $DdependsOn$, $newIDs$)

Inputs: WeightedDisjunction D, WeightedDisjunct $DdependsOn$, Boolean $newIDs$

Outputs: WeightedDisjunction D' *equivalent copy of E*\

1: **if** $copyCount(D) =$ **null then**
2: $copyCount(D) \leftarrow 0$
3: **for all** $E \in D.ChildElems$ **do**
4: *will update disjunction / branch identifier*\
5: $UpdateCopyNbr\big(E, copyCount(D)\big)$ *see Alg. 5.16, sec. 5.7.1*\
6: **end for**
7: **return** D *don't need to copy first time*\
8: **end if**
9: $copyCount(D) \leftarrow copyCount(D) + 1$
10: Let D' be a copy of D
11: $isACopy(D') \leftarrow$ **true**
12: **if** $newIDs =$ **true then**
13: $D'.ID \leftarrow$ new unique $WeightedDisjunction.ID$
14: **else**
15: $D'.ID \leftarrow D.ID$
16: **end if**
17: $D'.DepOnElem \leftarrow DdependsOn$
18: $D'.ChildElems \leftarrow \emptyset$ *will add fresh copies*\
19: **for all** $E \in D.ChildElems$ **do**
20: Let E' be a copy of E
21: **if** $newIDs =$ **true then**
22: $E'.ID \leftarrow$ a new unique $WeightedDisjunct.ID$
23: **else**
24: $E'.ID \leftarrow E.ID$
25: **end if**
26: $UpdateCopyNbr\big(E', copyCount(D.ID)\big)$ *see Alg. 5.16, sec. 5.7.1*\
27: $isACopy(E') \leftarrow$ **true**
28: $copyAndAssociateChildDisjs(D') \leftarrow$ **false** *default*\
29: $E'.DepOnDisj \leftarrow D'$
30: **end for**
31: **return** D'

to be made (i.e. $copyCount(D)$ is one or more), then a new copy D' is be generated. The property $isACopy(D')$, which we defined earlier in this section, is set to *true* to indicate that D' is a copy. This is used by *AddType* to indicate that any disjunctions which depend on the disjunction D, will need to be be copied and updated to depend on D', when they are encountered. They are only copied if encountered to avoid this processing in the event that reasoning

is stopped earlier, due to constraints such as time or resources. The disjunction copy D' is set to depend on the weighted disjunct element $DdependsOn$, which was passed as input.

The algorithm will then make a copy all disjunct element E members of D, add these copies as child elements for the copy D'. It will also set the copies to depend on D'. If any disjunct element has any child disjunctions $E.ChildDisjs$, then the copy E' will refer to the same / original child disjunctions. These child disjunctions will be copied and dependencies updated if any E' is encountered by the $AddType$ algorithm, which we described earlier in this section. In addition, $copyAndAssociateChildDisjs(D')$ is initialised to $false$, and can only be set to $true$ be the $\forall$-rule. The copy index for each E' is updated using the $UpdateCopyNbr$ function as described earlier. We will now illustrate the process performed by the algorithms presented in this section, using an example. This example is illustrated in Figure 5.8. Suppose that all E represent weighted disjunct elements and that all D represent weighted disjunctions. Directed arrows from an D to a E means that D depends on E, and directed arrows from a E to an D means that E depends on D (i.e. E is a disjunct element of D).

In the figure, step a illustrates an initial disjunction hierarchy, established using Algorithm 5.3. Suppose there is a weighted disjunct E_1 containing a definition of the form $\forall R.D_1$ added to an individual x_0 which connects to two separate individuals x_1 and x_2 using the role R. D_1 is a disjunction such that $D_1 \equiv E_2 \sqcup E_3$ and $E_2 \equiv D_2$ where $D_2 \equiv E_4 \sqcup E_5$. In step b, the $\forall$-rule is applied to $\forall R.D_1$, which will add D_1 as a type for x_1 and x_2. However, D_1 is marked for copying by setting $copyAndAssociateChildDisjs(D_1.DepOnElem)$ to $true$. When the reasoner attempts to add D_1 as a type to the second individual x_2, after having already added D_1 to x_1, a copy D_1' is generated to be added to x_2. $CopyDisjunct$ in Algorithm 5.6 generates D_1', such that $D_1.ID = D_1'.ID$ and both D_1 and D_1' both depend on E_1. In this step, the disjunct members E_2 and

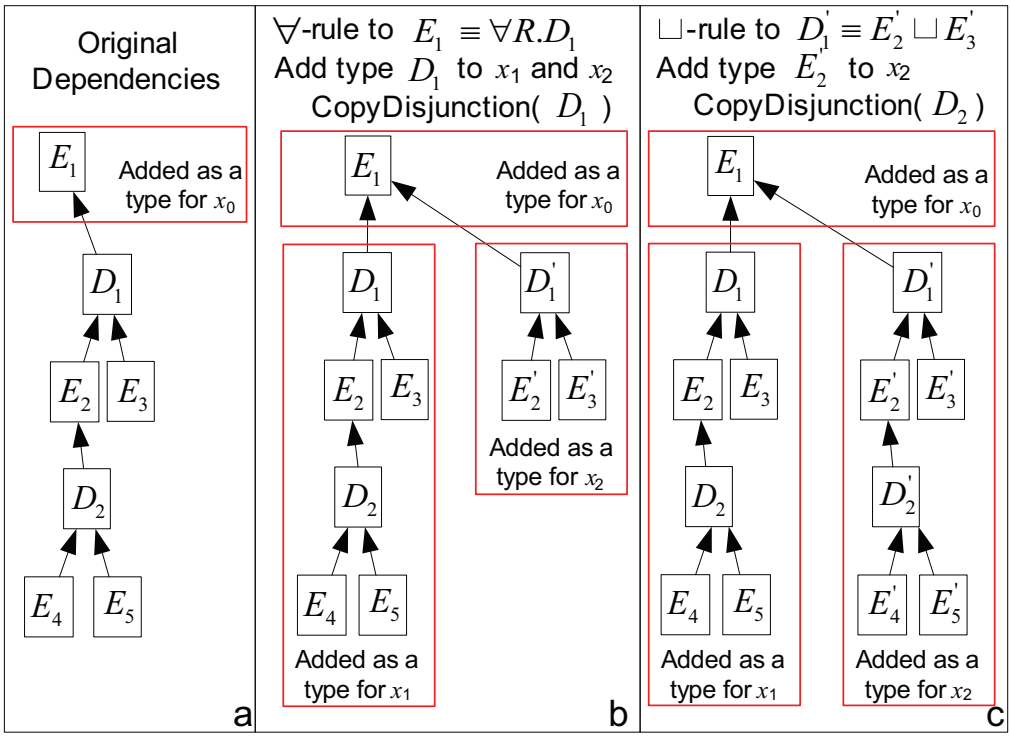


Figure 5.8: Example Copy Weighted Disjunction Hierarchy

E_3 of D_1 are also copied to generate E_2' and E_3' which depend on D_1'. However the dependencies for E_2' will not be updated until E_2' is added as a type to an individual. At this stage, E_2' has a reference to the original child disjunction D_2 (i.e. $E_2'.ChildDisjs = D_2$. Copied disjunct elements are marked as copies such that $isACopy(E_2')$ and $isACopy(E_3')$ are set to $true$, which will indicate that dependencies will need updating when they are encountered. In step c assume that the $\sqcup$-rule has been applied to the disjunction D_1' which is a type for x_2, causing E_2' to be expanded and added as a type to x_2. E_2' still refers to D_2 as its child disjunction, but since E_2' is marked as a copy, D_2 and its disjunct elements E_4 and E_5 are copied by $CopyDisjunct$ to generate D_2', E_4' and E_5'. The new disjunction copy D_2' is then set to depend on E_2'.

Now that we have defined the object hierarchy of dependencies between weighted disjunctions and disjunct element members, we will use this notation throughout the rest of this chapter. In the next section, we will detail the

process by which weights are associated with weighted disjunctions, and in particular how relative and normalised wright values are generated.

5.5 Weight Establishment

One of the key elements of our adaptive inference strategy is that we allow the user to assign weight values of importance to each requirement in the request. Each requirement in the request is as a conjunct element member of a conjunction or sub-conjunction in the request. As discussed in Section 3.4.2 Tableaux proves an inference by negation. Therefore, a conjunction becomes a disjunction. As such, in this section we will assume that every disjunct member which is derived from the user request, has an explicitly user specified weight value w associated with it, where $0 \leq w \leq 1$. An explicit weight value of 1 is the strongest level of importance while a weight value of 0 is the weakest level of importance. In practise users can enter weights as a value between 1 to 10 which is converted into a w value. Alternatively, request condition priorities can also be specified as high/medium/low which may correspond to weight values of 1/0.5/0.1 or even mandatory/non-mandatory 1/0.1. Alternatively, weights may be gathered implicitly using historical and user profile / preference data (Jung, 2006; desJardins et al., 2006). For instance, if half of the services which a mobile user has previously found, are close in proximity to the user, this a feature could be inserted into future requests with a weight of 0.5. However, user input mechanisms are not the focus of this thesis. Therefore, we will assume a value w has been associated with each condition in the request.

In addition to the explicit weight w, we also define relative and normalised relative weights, which are generated by our adaptive inference strategy, based on the explicit weight w. Only disjunct elements, i.e. instances of $Weighted\text{-}Disjunct$ defined in Section 5.4, have weights associated with them. Let $w(E_i)$ denote an explicit weight, $rw(E_i)$ denote a relative weight, $nrw(E_i)$ denote a

normalised relative weight, where E_i is an instance of $WeightedDisjunct$, $0 \leq w(E_i) \leq 1$, $0 \leq rw(E_i)$ and $0 \leq nrw(E_i) \leq 1$.

Relative weight rw is used to define the order in which particular conditions in the request should be matched against the service description. A relative weight value for a particular disjunct E_i is offset by the relative weight of the parent disjunct E_p, where $E_p = E_i.DepOnDisj.DepOnElem$. The relative weight of E_i is calculated by Equation 5.1. If there is no E_p, implying that E_i is the top most element, then $rw(E_i) = w(E_i)$. Generating relative weights ensures that a particular disjunct element E_i can never be applied before the disjunction D which it is a member of.

$$\begin{aligned} rw(E_i) \;&=\; w(E_i) \;\times\; rw(E_p) \\ &\text{where } E_p = E_i.DepOnDisj.DepOnElem \end{aligned} \tag{5.1}$$

Relative weights are established whenever an explicit weight is added to a concept using Algorithm 5.7, where the first parameter E_i denotes the weighted disjunct representing the negated condition in the request which a weight is being associated with and v is the explicit user specified weight.

Algorithm 5.7 AddExplicitWeight(E_i, v)

Inputs: WeightedDisjunct E_i, Decimal v
Pre-conditions: $w(E_i) =$ **null**

1: $w(E_i) \leftarrow v$
2: $rw(E_i) \leftarrow v$
3: Let $E_p \leftarrow E_i.DepOnDisj.DepOnElem$
4: **while** $E_p \neq$ **null do**
5: Let $rwCurrDisj \leftarrow \sum_{E_c \in E_i.DepOnDisj.ChildElems} rw(E_c)$
6: **if** $rw(E_p) < rwCurrDisj$ **then**
7: $v \leftarrow rwCurrDisj - rw(E_p)$
8: $rw(E_p) \leftarrow rw(E_p) + v$
9: $E_p \leftarrow E_p.DepOnDisj.DepOnElem$
10: **else**
11: break *break out of loop*\
12: **end if**
13: **end while**

The algorithm sets v to the current explicit $w(E_i)$ weight and relative weight $rw(E_i)$ associated with E_i. It then adds v to the relative weight for all parents disjunct elements of E_i. However, a disjunct element, may have multiple child disjunctions in the case that the disjunct element is a conjunction containing more than one disjunction. Under standard Tableaux (see Section 3.4.2), if any conjunct member of a conjunction generates a clash, then the others conjuncts do not need to be evaluated. Therefore, in our algorithm, when we add v to the relative weight of a particular disjunct element E_i, which is a member of the disjunction D, then we only increase the relative weight of the parent element E_p, upon which D depends, by the difference between the sum of the relative weight of all disjunct members of D and the relative weight of E_p. In the case that the parent disjunct E_p, only has one child disjunction, then this increase will be the full amount of v. However, where E_p has multiple child disjunctions, it is possible that the disjunct members of the other child disjunctions have already been increased, thus E_p is already increased and no further increase to E_p is required.

For example, assume the user request for WiFi Internet or desktop Internet, which gives rise to the negated definition $\neg\mathsf{Request} \equiv (\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet}) \sqcap (\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet})$, where $w(\mathsf{WiFi}) = 0.5$, $w(\mathsf{Internet}) = 0.7$ in the first disjunction and $w(\mathsf{Desktop}) = 0.3$, $w(\mathsf{Internet}) = 0.9$ in the second disjunction. Assume that all weights have been added using Algorithm 5.7 except for Desktop. When Desktop is added this results in the sum of disjunct elements of the disjunction $(\neg\mathsf{Desktop} \sqcup \neg\mathsf{Internet})$, equalling 1.2. However, the parent element $\neg\mathsf{Request}$ already has a relative weight of 1.2, such that $rw(\neg\mathsf{Request}) = 1.2$, because the weights for other disjunction $(\neg\mathsf{WiFi} \sqcup \neg\mathsf{Internet})$ have already been added. Therefore, no further increase to $w(\neg\mathsf{Request})$ occurs.

Normalised relative weight nrw is used to calculate the current degree of match between a user request and a service description at any time during the matching process. Degree of match is described in Section 5.8. Our normalised

relative weights imply that the sum of all nrw values of the disjunct elements E_i of a particular weighted disjunction D is equal to the nrw of the parent disjunct element E_p, where $E_p = E_i.DepOnDisj.DepOnElem$, such that Equation 5.2 is true for any E_p. The sum of the nrw values for all leaf weighted disjuncts E_i is equal to 1.0, where a weighted disjunct is considered a leaf if it does not contain any further weighted child disjunctions, such that $E_i.ChildElems = \emptyset$.

$$nrw(E_p) = \sum_{E_i \in D_y.ChildElems} nrw(E_i) \quad \text{where } D_y \in E_p.ChildDisjs \tag{5.2}$$

Normalised relative weights are generated using a call to Algorithm 5.8, after all the explicit weights $w(E_i)$, and thus relative weights $rw(E_i)$ have been set.

Algorithm 5.8 NormaliseWeights(E_p)

Inputs: WeightedDisjunct E_p
Pre-conditions: $nrw(E_p) = 1.0$ iff E_p is the top most disjunct

```
1: for all D_u ∈ E_p.ChildDisjs do
2:    Let S = Σ_{E_i ∈ D_u.ChildElems} rw(E_i)
3:    Let F ← S/nrw(E_p)
4:    for all E_i ∈ D_u.ChildElems do
5:       nrw(E_i) ← rw(E_i)/F
6:       NormaliseWeights(E_i)
7:    end for
8: end for
```

When the algorithm is first called the weighted disjunct element E_p containing the user request is passed to it. The request will have a total nrw value of 1.0, because 1.0 indicates a fully matching degree of match. The algorithm loops any child disjunctions D_u of the disjunct element E_p. For each of these D_u, let F denote the sum of the $rw(E_i)$ values for all disjunct elements E_i of D_u divided by the $nrw(E_p)$ of E_p. The $nrw(E_i)$ value for each disjunct element E_i of D_u is then the $rw(E_i)$ value of the this element divided by F. The algorithm then recursively repeats the process for each of these disjunct elements

to propagate the normalised result values down to all of the leaf elements. It is noteworthy that if there are multiple child disjunctions $E_p.ChildDisjs$ for a disjunct element E_p, this means that the disjunct element E_p is a conjunction containing more than one sub-disjunctions. In this case, all disjunctions in $E_p.ChildDisjs$ depend on E_p and only one of these disjunctions needs to generate a clash for all expansions in order to generate a clash for E_p. Therefore, the nrw value of E_p is not divided if there are multiple child disjunctions D_u. Rather, the sum the relative normalised weight values, for the disjunct element members of each of these child disjunctions D_u, is the same.

Going back to the example from earlier in this section, using algorithm 5.8 to generate normalised weights, would result in $nrw(\mathsf{WiFi}) = 0.42$, $nrw(\mathsf{Internet}) = 0.58$ in the first disjunction and $nrw(\mathsf{Desktop}) = 0.25$, $nrw(\mathsf{Internet}) = 0.75$ in the second disjunction. Assume that the service description met the desktop Internet requirement, but not the WiFi Internet requirement, the end result is still a degree of match (see Section 5.8) of 1.0, by adding 0.42 and 0.58, because the user only required desktop Internet or WiFi Internet, not both.

Now that we have defined the weights associated with weighted disjunct elements which represent conditions in the user request, we will use these weights in the next section to help influence Tableaux expansions. In particular, the weights are used to assist in determining which disjunction to apply the $\sqcup$-rule to next, in our adaptive inference strategy.

5.6 Disjunction Selection and Application

In this section we will describe the algorithms and the structures which control the order in which disjunctions are applied, giving rise to tree expansion. In Section 5.6.1 we will first provide an overview of our modified $\sqcup$-rule, for our proposed adaptive inference strategy. In Sections 5.6.2 and 5.6.3 we describe the algorithms and mechanisms for selecting which disjunction to apply the $\sqcup$-rule to next. In particular Section 5.6.2 discusses the various queues which we

used to order weighted disjunctions and Section 5.6.3 discusses the algorithm which uses these queues to select the next disjunction based on priorities.

5.6.1 Disjunction Rule Application

As discussed previously, one of the key objectives of our adaptive inference strategy is to prioritise the order in which the conditions in the user request are matched against the service description. Different conditions are represented as conjunctions and sub-conjunctions and since Tableaux proves or disproves an inference by negation, these become disjunctions and child disjunctions. In this section we discuss our modifications to the $\sqcup$-rule application to disjunctions. As discussed previously in Section 3.5.1 the $\sqcup$-rule gives rise to expansions controlled using branches in an expansion tree. This expansion occurs in depth-first / arbitrary order using standard Tableaux without taking into account the priority importance of these features to the user. In our adaptive inference strategies, disjunctions derived from the user request are given weight values. As discussed in Section 5.4, we call these weighted disjunctions and call all other disjunctions occurring in the ontology, non-weighted disjunctions. In our adaptive strategy tree expansion occurs in weighted disjunction order.

The main difference in our approach is that the $\sqcup$-rule creates a new branch node b_k in the expansion tree $\mathcal{G}$ for every disjunct element member of a disjunction which is applied, whereas standard Tableaux creates a branch point node b_k for each disjunction (rather than a disjunct element).

We do this to allow the reasoner to jump between simultaneously open branches by selecting a particular branch node b_k to jump to. As we mentioned in Section 5.3.1, in our adaptive strategy the $\sqcup$-rule jumps to different branch node depending on which weighted disjunct element is being applied next. The $\sqcup$-rule is re-applied for every weighted disjunct element member of every weighted disjunction. If a clash occurs and the clashing concept depends on a weighted disjunct element, then a backjump and reasoner restore

(which discards branches and assertions in standard Tableaux) does not occur. Rather, the $\sqcup$-rule is applied to the next applicable disjunct element of a weighted disjunction according to its weight, and our propose strategy will move to the relevant branch b_k without discarding branches. This functionality for handling weighted disjunctions is reflected in Algorithm 5.9 which illustrates the $\sqcup$-rule, which has been modified to enable our adaptive inference strategy.

Algorithm 5.9 AdaptApplyDisj($\mathcal{A}$, $\mathcal{G}$)

Inputs: ABox $\mathcal{A}$, CompletionGraph $\mathcal{G}$
Outputs: ClassConcept *clash*

Let $D(x) \leftarrow GetNextDisj(\mathcal{A}, \mathcal{G})$ *see Algorithm 5.12, Section 5.6.3*\
if D is an instance of $WeightedDisjunction$ **then**
  Let $E \leftarrow NextElem(D)$
  Let b_k be newly created $BranchNode$
  $\mathcal{G} \leftarrow \mathcal{G} \cup \{b_k\}$
  $\mathcal{G}.activeAdapBranch \leftarrow b_k$
  $createdByAssertion(b_k) \leftarrow E(x)$
  $depOnAdapBranch(b_k) \leftarrow depOnAdapBranch(D.DepOnElem)$
  $depOnAdapBranch(E) \leftarrow b_k$
  $AssignAdapBranchID(b_k, E)$ *see Algorithm 5.17, Section 5.7.1 *\
  $appliedConcept(E) \leftarrow$ **true** *set by Algorithm 5.9 Section 5.6.1*\
  $AddType(E, x, \mathcal{A})$ *do $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{E\}$, see Alg. 5.5 Sec. 5.4*\
  if $\{E, \neg E\} \subseteq \mathcal{L}(x)$ **then**
    return E * immediate clash *\
  end if
else
  apply standard Tableaux $\sqcup$-rule\
  Let $clash \leftarrow ApplyDisjRules\big(D(x), \mathcal{A}, \mathcal{G}\big)$ *see Alg. 3.2 Sec. 3.5.1*\
  Let b_k be the new BranchNode created by $ApplyDisjRules$
  $adapBranchID(b_k) \leftarrow adapBranchID(\mathcal{G}.activeAdapBranch)$
  return *clash*
end if
return null

In the algorithm, the function $GetNextDisj$, which is defined in Algorithm 5.12 in Section 5.6.3, is called to obtain the next assertion $D(x)$ to apply the $\sqcup$-rule to. Obviously this assertion will be a disjunction D that has been asserted as a type for a particular individual x. If D is a weighted disjunction, then the next disjunct element to apply is obtained using a call to $NextElem(D)$

(presented in Section 5.6.2). This function returns the disjunct element member of D which has not yet been applied and has the highest relative weight. A new branch node b_k is created and this is added to the expansion tree $\mathcal{G}$. In addition, b_k is set to be the new active adaptive inference branch node in the expansion tree, where $\mathcal{G}.activeAdapBranch$ denotes this active branch node. This is set because our adaptive inference strategy may have multiple simultaneously unfinished branches in the tree and the ABox state will reflect the currently active branch. This will be explained further in Section 5.7.

The $\sqcup$-rule then associates branch dependencies used by our adaptive inference strategy. Let $createdByAssertion(b_k)$ denote the assertion $E(x)$ which when applied, resulted in the creation of the new branch node b_k. We refer to a branch node created due to the application of a weighted disjunction, as an adaptive branch node. Alternatively, we call a branch created by application of a standard / non-weighted disjunction, a standard branch node. In Section 3.5.1 we defined $depOnBranch(w)$, which denotes the standard branch node which a class concept w, or role w, depends on. We record the last adaptive branch dependency separately. Let $depOnAdapBranch(w)$ denote the last adaptive branch node which w depends on, where w is a class concept C, role R or another branch node b_k. In the algorithm, the new adaptive branch node b_k is set to depend on the branch node which P depends on, where P is the parent element of E, such that $P = E.DepOnElem$, using the weighted disjunction notation defined in Section 5.4. In addition, the algorithm calls $AssignAdapBranchID(b_k, E)$, which assigns a unique identifier with the new branch node b_k. $AssignAdapBranchID$ is discussed in Section 5.7.1. Let $appliedConcept(E)$ denote whether or not the weighted disjunct element E has been applied. This is set to *true* by the algorithm for any E which is applied by the $\sqcup$-rule. This is used in Section 5.6.2 to determine the next disjunct element of a disjunction D to apply. Finally the algorithm adds the

weighted disjunct E as a type to the individual x as in standard Tableaux, and if this generates a clash, the clashing concept is returned.

If the disjunction D being applied is a non-weighted disjunction then it is applied in the standard way, using a standard call to $ApplyDisjRules$, which we defined in Algorithm 3.2, Section 3.5.1. However, as will be outlined in Section 5.6.3 non-weighted disjunctions are only applied after all weighted disjunctions have been applied. When applying non-weighted disjunctions, the only difference with our adaptive strategy, compared to standard Tableaux, is that non-weighted disjunctions C are also associated with an adaptive strategy branch identifier $depOnAdapBranch(C)$. This is because in the case that application of a non-weighted disjunction generates a clash for all expansions, the adaptive inference strategy will need to determine which weighted disjunct this depends on, in order to update degree of match.

In this section we provided an overview of our modified approach to the $\sqcup$-rule to incorporate our adaptive inference strategy. This modified $\sqcup$-rule utilises an algorithm called $GetNextDisj$ to obtain the next disjunction to apply. This algorithm utilises queues to determine which disjunction to apply next. The queues will be explained in the next section, before the $GetNextDisj$ algorithm itself in Section 5.6.3.

5.6.2 Disjunction Ordering and Queues

In this section we define iterators and queues which establish ordering of weighted disjunctions. Firstly, we define the $NextElem(D)$ function, in Equation 5.3 which returns the next disjunct element of a weighted disjunction D, which has not yet been applied by the $\sqcup$-rule. This is contrary to current Tableaux where the next disjunct element to apply is usually selected in the order that it appears or using some other heuristic but does not take importance to the user into account.

$$NextElem(D) = E \quad \text{such that}$$
$$max\Big(rw(E)\Big) \quad \text{where } E \in D.ChildElems \quad \text{and} \quad (5.3)$$
$$appliedConcept(E) = false$$

Equation 5.3 returns the weighted disjunct element contained in $D.ChildElems$, which has both the highest relative weight and $applied - Concept(E)$ set to $false$, where $appliedConcept(E)$ is set by the $\sqcup$-rule when an E is applied in Algorithm 5.9 in Section 5.6.1.

Now we establish an iterator and several queues which will be used to determine which disjunction which the $\sqcup$-rule applies next. These queues replace the $ToDo$ list of standard Tableaux (see Section 3.4.2), which is used to determine which assertions to apply the transformation rules to. However, we must first consider some factors which influence disjunction application order:

- A disjunction can only be applied (or re-applied) after it has been added as a type to an individual in the ABox $\mathcal{A}$. For instance, if a negated request is of the form ($\neg$InternetRequest $\sqcup$ $\neg$Cafe) and $\neg$ InternetRequest $\equiv$ ($\neg$WiFi $\sqcup$ $\neg$Internet), then the disjunct element $\neg$InternetRequest must be applied, before ($\neg$WiFi $\sqcup$ $\neg$Internet) is applicable;

- As we have stated in Section 5.1, our adaptive inference strategy continues matching requirements in the user request even if some condition fails, only stopping when other constraints such as resources or time are exhausted. Since conjunctive conditions in the request become disjunctions when negated, expansion of weighted disjunctions must continue under our strategy, even if one branch expansion failed to generate a clash;

- As described in Section 3.4.2 and in Section 5.4, in the case that the $\forall$-rule is applied to a universal quantifier which contains a disjunction in

its role filler, such as $\forall R.(E_1 \sqcup E_2)$ which is a type for an individual x_r, then a copy of $E_1 \sqcup E_2$ is added to all individuals x_p, $1 \leq p \leq m$, which x_r connects to using role R. Recall from Section 5.4 that these disjunction D copies share the same unique identifier $D.ID$. These copied disjunctions $E_1 \sqcup E_2$ depend on $\forall R.(E_1 \sqcup E_2)$, meaning that only one disjunction needs to generate a clash for all expansions in order to clash for $\forall R.(E_1 \sqcup E_2)$. Therefore, in the case that E_1 is a weighted disjunct which generated a clash, then E_2 of the same disjunction copy should be applied before E_1 of another copy of the disjunction $E_1 \sqcup E_2$, even through E_2 has a lower weight than E_1;

- Non-weighted disjunctions are only applied after no more weighted disjunctions can be applied for a particular expansion.

In order to control which weighted disjunction to apply next we establish an iterator and due to the considerations described above we establish three separate queues for weighted disjunctions and a fourth queue for non-weighted disjunctions:

- Let $DisjElemIter$ be an iterator which contains all weighted disjunct element E_i members from weighted disjunctions which have been added as types to individuals;

- Let the following queues contain weighted disjunctions D which have been added as types to individuals, but have not yet had all of their disjunct elements applied, such that:

  $\mathcal{CQ}_u$ contains a weighted disjunction D where no disjunct element E_i member of D has been applied yet;

  $\mathcal{CQ}_{pa}$ contains a weighted disjunction D where at least one disjunct element E_i member of weighted disjunction D has been applied and all E_i which have been applied have also been found to generate a clash;

$\mathcal{CQ}_{pn}$ contains a weighted disjunction D where at least one disjunct element E_i member of D has been applied but at least one of the applied E_i did not yet generate a clash.

- Let $\mathcal{DQ}$ denote a queue which contains all other non-weighted disjunctions which have been added as a type to an individual and have not yet been applied. Recall, non-weighted disjunctions are those which were not derived from the user request and they are applied in the standard Tableaux way.

We store weighted disjunctions in separate queues from non-weighted disjunctions because of our preference to apply non-weighted disjunctions only after all weighted disjunctions have been applied for an expansion. The three separate weighted disjunction queues arise from the fact that several equivalent copies of the same disjunction can exist due to the application of the $\forall$-rule as described earlier in this section. In the case that a disjunct element fails to clash for a particular weighted disjunction copy, then another copy of the same disjunction in the un-applied disjunctions queue will be evaluated next. In the case that all copies of the same weighted disjunction fail to generate a clash for at least one disjunct element, the strategy returns to partially evaluated disjunctions to continue checking these, due to our support for partial matching. In addition, a particular copy of a disjunction where all evaluated disjunct elements have clashed, is favoured over a copy of the same disjunction where all applied elements have not yet clashed, giving rise to the need for two separate partially applied queues. We will explain these priorities in more detail in the next section. The remainder of this section concentrates on defining the structure of our iterator and queues.

The structure of iterator $DisjElemIter$ is defined in Equation 5.4.

$$
\begin{aligned}
DisjElemIter \; = \{E_1, ..., E_n \mid (1 \leq i \leq n)\} \quad & \text{where} \\
& E_i \;\; \text{is an instance of WeightedDisjunct} \quad \text{and} \\
& rw(E_i) \geq rw(E_j), \quad 1 \leq i \leq j \leq n, \quad \text{and} \\
& \text{no } E_i.ID \text{ occurs more than once in } ElemElemIter
\end{aligned}
\tag{5.4}
$$

The iterator $DisjElemIter$ will be used to control which weighted disjunction to apply next. It is not used for and does not contain non-weighted disjunctions. Rather $DisjElemIter$ contains all the unique weighted disjunct element E_i members of all the weighted disjunction identifiers $D.ID$ added as types to individuals in the knowledge base's ABox $\mathcal{A}$. That is, if the $\forall$-rule generates adds multiple copies of the same disjunction sharing the same $D.ID$, only one copy of the disjunct elements from D are added to $DisjElemIter$. However, as we will explain in Section 5.6.3, these elements will remain in the queue until either all copies of D have been evaluated, or a D has generated a clash for all expansions. The $DisjElemIter$ is ordered in descending $rw(E_i)$ order. Let $DisjElemIter.NextElem$ denote a property that holds a disjunct element which is contained in $DisjElemIter$, such that $DisjElemIter.NextElem \in DisjElemIter$. This element will be used to maintain the current iteration of the iterator $DisjElemIter$ and the algorithm which controls this, will be described in the Section 5.6.3. Having defined $DisjElemIter$, we now define each of our four queues. First we will define the queues which hold weighted disjunctions $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$, then we will define the queue for non-weighted disjunctions $\mathcal{DQ}$.

As described earlier in this section, the queues $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$ contain weighted disjunctions. A particular weighted disjunction is added to only one of these queues depending on its current state. The iterator $DisjElemIter$ defined above, will be used to obtain a particular unique identifier $D.ID$ of a weighted disjunction D (see Section 5.4) to apply next, which will be explained

in Section 5.6.3. Given a $D_i.ID$, a specific assertion $D_j(x_r)$ about a weighted disjunction D_j copy which has the identifier $D_i.ID$, will be selected by the queues, such that $D_j.ID = D_i.ID$. This motivates the structure of these three queues $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$, which are made up of disjunction identifier $D_i.ID$ and $DSet_i$ pairs. Let $DSet_i$ be a set containing assertions $D_j(x_r)$ about equivalent copies of a weighted disjunction D_i and where x_r is an individual to which D_j has been added as a type, such that $D_i.ID = D_j.ID$, where $\{D_i(x_r), D_j(x_p)\} \subseteq DSet$, $1 \leq i \leq j \leq n$, $1 \leq r \leq p \leq q$.

We now define each of the weighted disjunction queues. The structure of $\mathcal{CQ}_u$ which contains weighted disjunctions where no disjunct elements have been applied, is presented in Equation 5.5.

$$\begin{aligned} \mathcal{CQ}_u \;=\; & \Big\{\langle D_1.ID, DSet_1\rangle, ..., \langle D_n.ID, DSet_n\rangle\Big\}, \;\; 1 \leq i \leq n \quad \text{where} \\ & DSet_i = \Big\{D_1(x_1), ..., D_m(x_q)\Big\}, \;\; 1 \leq j \leq m, \; 1 \leq r \leq q \\ & D_j \text{ is an instance of } WeightedDisjunction, \; x_r \text{ is an individual} \\ & \text{every } \langle D_i.ID, DSet_i\rangle \text{ pair has a distinct } D_i.ID \text{ value} \end{aligned} \tag{5.5}$$

In $\mathcal{CQ}_u$, the weighted disjunction assertions in the set $DSet$ are un-ordered. However, if the caching strategy CS (see Section 4.5) is used in conjunction with our adaptive inference strategy, then an instance of the CS queue $\mathcal{Q}$ is used in place of $DSet$ for ordering un-applied disjunctions based on the cache.

The structure of $\mathcal{CQ}_{pa}$ which contains weighted disjunctions where at least one disjunct element has been applied and all applied elements have been found to clash, is given in Equation 5.6.

$$
\begin{aligned}
\mathcal{CQ}_{pa} \;=\; & \Big\{\langle D_1.ID, DSet_1\rangle, ..., \langle D_n.ID, DSet_n\rangle\Big\}, 1 \le i \le n \quad \text{where} \\
& DSet_i = \Big\{D_1(x_1), ..., D_m(x_q)\Big\}, \quad 1 \le j \le m, 1 \le r \le q \\
& D_j \text{ is an instance of } WeightedDisjunction, \quad x_r \text{ is an individual} \\
& rw\Big(NextElem(D_j)\Big) \le rw\Big(NextElem(D_k)\Big), \\
& \qquad \text{for all } D_j(x_r), D_k(x_p) \in DSet_i, 1 \le j \le k \le m \\
& \text{every } \langle D_i.ID, DSet_i\rangle \text{ pair has a distinct } D_i.ID \text{ value}
\end{aligned}
\tag{5.6}
$$

In $\mathcal{CQ}_{pa}$ the set $DSet$ contains weighted disjunction assertions which are in ascending order by relative weight of the next disjunct element of the disjunction to apply, where next element was defined earlier in this section. The reason the order is ascending is as follows. The $DSet$ contains all equivalent copies of a weighted disjunction added as types to different individuals and if any disjunction generates a clash for all disjunct elements then the others do not need to be checked. All disjunctions in this queue have generated a clash for all applied disjunct elements. Therefore, the goal is to select the disjunction with the fewest remaining disjunct elements to evaluate, in order to prove a clash for all elements. This is the disjunction with the lowest next applicable element relative weight.

The structure of $\mathcal{CQ}_{pn}$ which contains weighted disjunctions where at least one disjunct element has been applied but not all applied elements have been found to clash, is given in Equation 5.7.

$$
\begin{aligned}
\mathcal{CQ}_{pn} \;=\; & \Big\{\langle D_1.ID, DSet_1\rangle, ..., \langle D_n.ID, DSet_n\rangle\Big\}, \quad 1 \leq i \leq n \quad \text{where} \\
& DSet_i = \Big\{D_1(x_1), ..., D_m(x_q)\Big\}, \quad 1 \leq j \leq m, 1 \leq r \leq q \\
& D_j \text{ is a } WeightedDisjunction, \quad x_r \text{ is an individual} \\
& degMatch(D_j) \geq degMatch(D_k), \\
& \qquad \text{for all } D_j(x_r), D_k(x_p) \in DSet_i, 1 \leq j \leq k \leq m \\
& \text{every } \langle D_i.ID, DSet_i\rangle \text{ pair has a distinct } D_i.ID \text{ value}
\end{aligned}
\tag{5.7}
$$

In $\mathcal{CQ}_{pn}$ the set $DSet$ contains weighted disjunction assertions which are in descending order by the degree of match $degMatch(D)$ for the disjunction D. The degree of match is the sum of the normalised relative weight nrw for the disjunct elements of a disjunction which has been found to clash and will be presented in Section 5.8. For example, assume there is a weighted disjunction $\neg\mathsf{InternetRequest} \equiv (\neg\mathsf{Internet} \sqcup \neg\mathsf{WiFi} \sqcup \neg\mathsf{Service} \sqcup \neg\mathsf{FreeUse})$, where $nrw(\neg\mathsf{Internet}) = 0.4$, $nrw(\neg\mathsf{WiFi}) = 0.3$, $nrw(\neg\mathsf{Service}) = 0.2$ and $nrw(\neg\mathsf{FreeUse}) = 0.1$. Assume the disjunction is added as a type to the individuals PayInternet and FreeWiFiInternet due to application of a $\forall$-rule on $\forall$ sells.$\neg$InternetRequest, generating two equivalent copies. Assume the disjunction added to PayInternet generated a clash for $\neg$Internet giving a degree of match of 0.4 and the disjunction added to FreeWiFiInternet generated a clash for both $\neg$Internet and $\neg$WiFi giving a degree of match of $(0.4 + 0.3 = 0.7)$. When checking which of the two disjunctions to apply next, the disjunction added to FreeWiFiInternet should be applied first because it has a higher degree of match (i.e 0.7) than the one added to PayInternet (i.e. 0.4). A higher degree of match for a disjunction indicates that a larger number of its disjunct elements have been successfully matched so far.

Now that we have defined our three weighted disjunction queues, we will define functions which can be called on these queues:

- Let $\mathcal{CQ}.Contains(D_i.ID)$ be a function which can be used to determine whether or not the queue contains any more assertions about weighted disjunctions which have the unique identifier $D_i.ID$. As such, $\mathcal{CQ}.Contains(D_i.ID)$ returns *true* if $\langle D_i.ID, DSet_i \rangle \in \mathcal{CQ}$, $DSet \neq \emptyset$, or $false$ otherwise;

- Let $\mathcal{CQ}.GetNext(D_i.ID)$ be a function which obtains the next assertion about a weighted disjunction which has the unique identifier $D_i.ID$. This is defined as the assertion $D_1(x_1)$, where $D_1(x_1) \in DSet_i$ (i.e. $D_1(x_1)$ is the first assertion in $DSet_i$ according to ordering) and $\langle D_i.ID, DSet_i \rangle \in \mathcal{CQ}$;

- Let $\mathcal{CQ}.Add\Big(D_j(x_r)\Big)$ be a function which adds an assertion $D_j(x_r)$ to the queue. $D_j(x_r)$ is added to a $DSet_i$, such that $DSet_i \leftarrow DSet_i \cup \Big\{D_j(x_r)\Big\}$, for the $DSet_i$ where $\langle D_j.ID, DSet_i \rangle \in \mathcal{CQ}$. A new $DSet_i$ may be created if one does not already exist for $D_j.ID$;

- Let $\mathcal{CQ}.Remove\Big(D_j(x_r)\Big)$ be a function which removes an assertion $D_j(x_r)$ from the queue. An assertion $D_j(x_r)$ is removed from $DSet_i$, where $D_j(x_r) \in DSet_i$ and $\langle D_j.ID, DSet_i \rangle \in \mathcal{CQ}$.

- Let $\mathcal{CQ}.RemoveAll(D_i.ID)$ remove all assertions $D_j(x_r) \in DSet_i$ about the disjunctions D_j which have the unique identifier $D_i.ID$, from the queue, such that the pair $\langle D_i.ID, DSet_i \rangle$ (i.e. $D_j.ID = D_i.ID$);

- Let $\mathcal{CQ}.IsEmpty$ be a property which can be used to determine whether or not the queue contains anymore weighted disjunctions. It returns *true* if $\mathcal{CQ} = \emptyset$ or $DSet_i = \emptyset$ for all $\langle D_i.ID, DSet_i \rangle \in \mathcal{CQ}$, or $false$ otherwise.

This completes our discussion of the weighted disjunction queues. Now we define the queue $\mathcal{DQ}$ which contains non-weighted disjunctions, which were not derived from the user request. The structure of this queue is given in Equation 5.8.

$$\mathcal{DQ} = \Big\{D_1(x_1), ..., D_n(x_q)\Big\}, \quad 1 \leq j \leq m, 1 \leq r \leq q \quad \text{where}$$
$$D_i \text{ is a non-weighted disjunction of the form } D_j \equiv E_1 \sqcup ... \sqcup E_z$$
$$x_r \text{ is an individual} \tag{5.8}$$

$\mathcal{DQ}$ contains assertions about non-weighted disjunctions which have been added as types of individuals, which have not yet had the $\sqcup$-rule applied to them. The queue $\mathcal{DQ}$ is un-ordered. However, if the caching strategy CS (see Section 4.5) is used with our adaptive inference strategy, then an instance of the CS queue $\mathcal{Q}$ is used in place of $\mathcal{DQ}$ for ordering non-weighted disjunctions which have not been applied, according to the cache.

We have now defined the iterator and the queues used in our adaptive inference strategy to hold disjunctions. We will now illustrate the way in which disjunctions and their disjunct elements are added to their respective queue, and the disjunct element iterator, when they are added as a type to an individual. Let *AddType* be an algorithm which is called whenever a class concept C definition is asserted as a type to an individual x, shown in Algorithm 5.10.

Algorithm 5.10 AddType(C, x, $\mathcal{A}$)

Inputs: ClassConcept C, Individual x, ABox $\mathcal{A}$

1: **if** C is an instance of $WeightedDisjunction$ **then**
2: $\quad \mathcal{CQ}_u.Add\Big(C(x)\Big)$
3: $\quad$ **for all** $E_i \in C.ChildElems$ **do**
4: $\quad\quad$ **if** $E_j \notin DisjElemIter$ for any E_j where $E_j.ID = E_i.ID$ **then**
5: $\quad\quad\quad DisjElemIter \leftarrow DisjElemIter \cup \{E_i\}$
6: $\quad\quad$ **end if**
7: $\quad$ **end for**
8: $\quad$ **if** $rw(DisjElemIter.NextElem) > rw\Big(NextElem(C)\Big)$ **then**
9: $\quad\quad DisjElemIter.NextElem \leftarrow NextElem(C)$
10: $\quad$ **end if**
11: **else if** $C \equiv E_1 \sqcup ... \sqcup E_n$ **then**
12: $\quad \mathcal{DQ} \leftarrow \mathcal{DQ} \cup \{C\}$
13: **end if**
14: * add the type C to x, by calling $AddType(C, x, \mathcal{A})$ defined in Algorithm 5.5 in Section 5.4 *\

In the *AddType* algorithm, in the case that the class concept definition C being added as a type to the individual x is a weighted disjunction, it is added to the un-applied disjunction queue $\mathcal{CQ}_u$. In addition, all disjunct elements E_i of the disjunction C are added to the element iterator $DisjElemIter$ provided it does not already contain other elements which share the same $E_i.ID$. Furthermore, if the relative weight of the next disjunct element $NextElem(C)$ of the weighted disjunction C, is higher than the relative weight of the next element in the iterator $DisjElemIter.NextElem$, then $DisjElemIter.NextElem$ is set to $NextElem(C)$. This is because $DisjElemIter.NextElem$ is now applicable disjunct with the highest relative weight. Alternatively, in the case that C is a non-weighted disjunction, it is added to $\mathcal{DQ}$. The disjunction C is then added as a type for x, as in standard Tableaux. It is noteworthy, that we also defined an *AddType* procedure in Algorithm 5.5 in Section 5.4 for adding types to individuals. Algorithm 5.10 presented in this section can be assumed to be in addition to Algorithm 5.5. Therefore, Algorithm 5.10 calls Algorithm 5.5 on the last line, to add the type.

When a clash is detected during the inference check and this clash depends on a weighted disjunct element E_i, of a weighted disjunction D, then the $UpdateQueuesOnClash$ algorithm is called as was shown in Algorithm 5.2 in Section 5.3.1). We will now define $UpdateQueuesOnClash$ in Algorithm 5.11.

The reason for this algorithm is as follows. As described earlier in this section, we have separate queues for partially applied weighted disjunctions where not all disjunct elements have generated a clash $\mathcal{CQ}_{pn}$ compared to those where all applied elements have clashed $\mathcal{CQ}_{pa}$. When a disjunction in $\mathcal{CQ}_u$ is applied for the first time, it is moved to $\mathcal{CQ}_{pn}$ which contains partially applied disjunctions where applied elements have not yet generated a clash (this move will be described in the next section). When a clash occurs, the $UpdateQueuesOnClash$ algorithm will move it from $\mathcal{CQ}_{pn}$ to $\mathcal{CQ}_{pa}$ if all applied disjunct elements now clash. A weighted disjunct element E is known to

Algorithm 5.11 UpdateQueuesOnClash(E_i)

Pre-conditions: E_i generated a clash
Inputs: WeightedDisjunct E_i

1: $clashingConcept(E_i) \leftarrow$ **true**
2: **if** $clashingConcept(E_j) =$ **true** for all $E_j \in E_i.DepOnDisj.ChildElems$
where $appliedConcept(E_j) =$ **true** and
there exists some $E_j \in E_i.DepOnDisj.ChildElems$
where $appliedConcept(E_j) =$ **false then**
3: *all applied disjunct elements of $E_i.DepOnDisj$ now clash*\
4: $\mathcal{CQ}_{pn}.Remove(E_j.DepOnDisj)$
5: $\mathcal{CQ}_{pa}.Add(E_j.DepOnDisj)$
6: **else if** $clashingConcept(E_j) =$ **true** and
$appliedConcept(E_j) =$ **true**, for all $E_j \in E_i.DepOnDisj.ChildElems$ **then**
7: *if all elements clashing*\
8: $\mathcal{CQ}_{u}.RemoveAll(E_i.DepOnDisj.ID)$
9: $\mathcal{CQ}_{pa}.RemoveAll(E_i.DepOnDisj.ID)$
10: $\mathcal{CQ}_{pn}.RemoveAll(E_i.DepOnDisj.ID)$
11: $UpdateQueuesOnClash(E_i.DepOnDisj.DepOnElem)$
12: **end if**

have generated a clash iff $clashingConcept(E) = true$, which is set by Algorithm 5.2 in Section 5.3.1) as well as $UpdateQueuesOnClash$ itself. A disjunct element is known to have been applied iff $appliedConcept(E) = true$ which is set by Algorithm 5.9 in Section 5.6.1.

Alternatively, it may be the case that all disjunct element members of the disjunction D which E is a member, have now been applied and also clash. In this situation no further copies of D need to be applied. As described in Section 5.4, all disjunctions with the same $D.ID$ depend on the class concept / branch node, so if one of these disjunctions has generated a clash for all of its disjunct members, then the others do not need to be evaluated. Therefore, all equivalent copies of D sharing the same unique identifier $D.ID$ are removed from the queues. In addition, when all disjunct elements of a disjunction D are clashing, this means that the parent disjunct P for D has effectively generated a clash as well, because D depends on P. Therefore, $UpdateQueuesOnClash$ performs a recursive call back to update these parent disjunct elements.

In this section we have described the iterator and queues used by our adaptive inference strategy to store disjunctions. The iterator is a weight ordered place holder to determine which disjunct element to apply next. The queues provide ordering of weighted disjunctions in various states and another queue contains non-weighted disjunctions. These iterator and queues will be used in the next section, which describes the procedure for determining which disjunction to apply next and the considerations involved.

5.6.3 Disjunction Application Selection

In the previous section we detailed an iterator and queues which we used to order disjunctions which the $\sqcup$-rule is applicable to, by taking weight into account. In this section, we will provide the mechanisms which use these queues to select the next disjunction to apply the $\sqcup$-rule to. In addition, at the beginning of the previous section we identified the fact that the $\forall$-rule may create several copies of the same disjunction added as types to different individuals and if one of these copies generates a clash for all expansions then the others do not need to be evaluated, because they depend on the same universal quantifier definition. In standard Tableaux this implies that the reasoner evaluates a particular disjunction until a disjunct element fails to generate a clash, then it begins evaluating another copy of the same disjunction. However, since our strategy supports partial matching, the reasoner will need to return to these disjunctions in the case that no disjunction can be found where all disjunct elements generate a clash.

This gives rise to the following priorities for selecting which particular equivalent copy of the same disjunction (which is added as a type to a different individual) to apply next:

1. The highest priority is given to a weighted disjunction D copy in the queue $\mathcal{CQ}_{pa}$ which contains only weighted disjunctions where all of the applied disjunct element E_i members of D have also generated a clash;

2. Second highest priority is given to a weighted disjunction D in the queue $\mathcal{CQ}_u$, which contains only weighted disjunctions which have not yet been applied;

3. Third priority is given to a weighted disjunction D in the queue $\mathcal{CQ}_{pn}$ which contains weighted disjunctions where at least one applied disjunct element E_i member of D has failed to generate a clash;

4. The lowest priority will be given to disjunction D in the queue $\mathcal{DQ}$ which contains non-weighted disjunctions which have not yet been applied.

The reason that copies of the same weighted disjunction D in $\mathcal{CQ}_{pa}$ are given the greatest priority, is that these have the fewest remaining disjunct element E_i members to check in order to prove a clash for the entire disjunction. Weighted disjunction copies in $\mathcal{CQ}_u$ are favoured over those in $\mathcal{CQ}_{pn}$, because a disjunction which has not had any of its disjunct element E_i members applied, has a chance of generating a clash for every E_i, while disjunctions in $\mathcal{CQ}_{pn}$ do not. Non-weighted disjunctions in $\mathcal{DQ}$ have the least priority because they are not related to the user request and are not associated with a weight value. These non-weighted disjunctions, however, will be applied after all weighted disjunctions, if there are sufficient remaining time or resources, in order to preserve completeness.

The priorities will give rise to the following example of disjunction selection. Assume there is a weighted disjunction $\neg\mathsf{InternetRequest} \equiv (\neg\mathsf{Internet} \sqcup \neg\mathsf{WiFi} \sqcup \neg\mathsf{FreeUse})$. Assume the disjunction is added as a type to the individuals Coffee, PayInternet and FreeWiFiInternet due to application of a $\forall$-rule on $\forall$ sells.$\neg$InternetRequest, generating three equivalent copies. These copies will initially be added to the un-applied queue $\mathcal{CQ}_u$. Assume that the disjunction added to Coffee, was selected and the $\neg$Internet disjunct was applied did not generate a clash. Rather than continuing to evaluate the same disjunction,

another un-applied disjunction from $\mathcal{CQ}_u$ is evaluated. Assume the disjunction added as a type to PayInternet is applied and that this generates a clash for the disjunct element ¬Internet. Evaluation of the same disjunction then continues because the first applied element has generated a clash, however, assume the next element ¬WiFi fails to generate a clash. The last disjunction which is a type for FreeWiFiInternet is then evaluated, and assume that both ¬Internet and ¬WiFi clash. The last disjunct element ¬FreeUse is then evaluated and generates a clash thereby proving a clash for the whole disjunction ¬InternetRequest.

In the remainder of this section we will detail the $GetNextDisj$ algorithm and the sub-algorithms which it uses. This is used by the ⊔-rule to obtain the next disjunction to apply, which was provided in Algorithm 5.9 in Section 5.6.1. $GetNextDisj$ and its sub-algorithms take account of the queue priorities which we detailed earlier in this section. $GetNextDisj$ is provided in Algorithm 5.12.

Algorithm 5.12 GetNextDisj($DisjElemIter$, $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$, $\mathcal{DQ}$)

Inputs: Set $DisjElemIter$, CompletionQueue $\mathcal{CQ}_u$, CompletionQueue $\mathcal{CQ}_{pa}$, CompletionQueue $\mathcal{CQ}_{pn}$

Outputs: Assertion $D(x)$

1: Let $D_j(x_r) \leftarrow createdByAssertion(\mathcal{G}.activeAdapBranch)$
2: **if** $D_j.ChildDisjs = \emptyset$ and $\mathcal{DQ} \neq \emptyset$ **then**
3: *cannot expand the last weighted disjunction D_j any further*\
4: Let $D_1(x_1)$ denote $D_1(x_1) \in \mathcal{DQ}$ where $\mathcal{DQ} = \left\{D_1(x_1), ..., D_n(x_n)\right\}$
5: remove $D_1(x_1)$ from $\mathcal{DQ}$
6: **return** $D_1(x_1)$
7: **else if** ($\mathcal{CQ}_u.IsEmpty \neq$ **true** or $\mathcal{CQ}_{pa}.IsEmpty \neq$ **true** or $\mathcal{CQ}_{pn}.IsEmpty$) $\neq$ **true then**
8: **return** $GetNextWeightedDisj(DisjElemIter, \mathcal{CQ}_u, \mathcal{CQ}_{pa}, \mathcal{CQ}_{pn})$
9: **end if**

Algorithm 5.12 returns the next disjunction to apply as well as the individual which it has been asserted as a type of. Therefore, the output of the algorithm is an assertion of the form $D(x)$ where D is a disjunction which may be weighted or non-weighted and x is an individual.

The algorithm retrieves the last weighted disjunction which was applied. If this disjunction cannot be further expanded then a non-weighted disjunction is

returned. Alternatively, if the last applied weighted disjunction can be further expanded (i.e. has child disjunctions) or there are no non-weighted disjunctions to apply, then an assertion about another weighted disjunction is retrieved using a call from $GetNextWeightedDisj$, which is defined in Algorithm 5.13

Algorithm 5.13 GetNextWeightedDisj($DisjElemIter$, $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$)

Inputs: Set $DisjElemIter$, CompletionQueue $\mathcal{CQ}_u$, CompletionQueue $\mathcal{CQ}_{pa}$, CompletionQueue $\mathcal{CQ}_{pn}$

Outputs: Assertion $D(x)$

1: **while** $\mathcal{CQ}_u.IsEmpty \neq$ **true** or $\mathcal{CQ}_{pa}.IsEmpty \neq$ **true** or $\mathcal{CQ}_{pn}.IsEmpty \neq$ **true do**
2: Let $E_k \leftarrow DisjElemIter.NextElem$
3: Let $D_i \leftarrow E_k.DepOnDisj$
4: Let $D_j(x_p) \leftarrow TryGetFromQueue(E_k, \mathcal{CQ}_u, \mathcal{CQ}_{pa}, \mathcal{CQ}_{pn})$
5: **if** $\mathcal{CQ}_u.Contains(D_i.ID) \neq$ **true** and $\mathcal{CQ}_{pa}.Contains(D_i.ID) \neq$ **true** and $\mathcal{CQ}_{pn}.Contains(D_i.ID) \neq$ **true then**
6: $DisjElemIter.NextElem \leftarrow E_{k-1}$ *de-iterate*\
7: remove E_k from $DisjElemIter$
8: **end if**
9: **if** $D_j(x_p) \neq$ **null then**
10: $DisjElemIter.NextElem \leftarrow NextElem(D_j)$
11: **end if**
12: **if** there exists $E_{k+1} \in DisjElemIter$ **then**
13: $DisjElemIter.NextElem \leftarrow E_{k+1}$ *iterate*\
14: **end if**
15: **if** $D_j(x_p) \neq$ **null then**
16: **return** $D_j(x_p)$
17: **end if**
18: **end while**

The purpose of this algorithm is to control the iterator $DisjElemIter$ in order to obtain the next weighted disjunction to apply, from one of the weighted disjunction queues $\mathcal{CQ}_u, \mathcal{CQ}_{pa}, \mathcal{CQ}_{pn}$ (which were defined in Section 5.6.2). The mean idea is that next weighted disjunct element to apply, is obtained from the iterator $DisjElemIter.NextElem$, where this element is a member of a weighted disjunction D_i. An assertion $D_j(x_p)$ about a copy of D_i is obtained from one of the queues, such that $D_j.ID = D_i.ID$. Then the iterator is moved to its next element and the assertion $D_j(x_p)$ is then returned, to be applied.

We will now explain the algorithm is more detail. Algorithm 5.13 loops while any of the weighted queues $\mathcal{CQ}_u, \mathcal{CQ}_{pa}, \mathcal{CQ}_{pn}$ are not empty and until an

assertion is found. The next weighted disjunct element E_k is obtained from the iterator. E_k is a member of which is a disjunct member of a weighted disjunction D_i. The disjunction D_i is used to obtain a specific assertion of the form $D_j(x_p)$ and remove it from the queues, using a call to $TryGetFromQueue$ (which is outlined in Algorithm 5.14 later in this section), where D_j is an equivalent copy of D_i such that $D_i.ID = D_j.ID$. $TryGetFromQueue$ may fail to obtain an assertion (e.g. the last copy of D_i may have been removed from the queues because it generated a clash for all elements). If there are no longer any assertions in the queues about a copy of D_i, then the current element E_k is removed from the iterator $DisjElemIter$ and the iterator is moved to the previous element. If $TryGetFromQueue$ obtained an assertion $D_j(x_p)$, the next element in the iterator $DisjElemIter$ is set to be the next element to apply D_j (i.e. this may be different E_k). For instance, if the next disjunct element in the iterator is E_1 which is a member of $D_j \equiv E_1 \sqcup E_2$ and $TryGetFromQueue$ finds that there an assertion $D_j(x_p)$ where E_1 has already generated a clash and E_2 has not yet been applied, then this assertion $D_j(x_p)$ will be selected over some other assertion $D_j(x_{p+1})$ where neither disjunct element has been applied. Therefore, the iterator is changed to reflect that E_2 is the next applied element which will be applied. Then $DisjElemIter$ iterates to the next element. The assertion $D_j(x_p)$ obtained by $TryGetFromQueue$ is then returned, so that the $\sqcup$-rule can be applied to it.

We will now discuss the algorithm $TryGetFromQueue$ which is passed the current element E_i in the iterator, and the weighted disjunction queues $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$ and $\mathcal{CQ}_{pn}$. This procedure is defined in Algorithm 5.14.

The purpose of the algorithm is to obtain an assertion $D_j(x_p)$ where D_j is an equivalent copy of a disjunction D_z which E_i is a member of, such that $D_z.ID = D_j.ID$. The algorithm takes the priorities for selecting which order to check each queues into consideration, which we discussed at the beginning of this section. First the $\mathcal{CQ}_{pa}$ queue is checked, which contains disjunctions

Algorithm 5.14 TryGetFromQueue(E_i, $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$, $\mathcal{CQ}_{pn}$)

Inputs: WeightedDisjunt E_i, Queue $\mathcal{CQ}_u$, Queue $\mathcal{CQ}_{pa}$, Queue $\mathcal{CQ}_{pn}$
Outputs: Assertion $D_j(x_p)$ where D_j is a weighted disjunction

1: **if** $\mathcal{CQ}_{pa}.Contains(E_i.DepOnDisj.ID)$ **then**
2: Let $D_j(x_p) \leftarrow \mathcal{CQ}_{pa}.GetNext(E_i.DepOnDisj.ID)$
3: **if** $rw\Big(NextElem(D_j)\Big) \geq rw(E_i.ID)$ **then**
4: $\mathcal{CQ}_{pa}.Remove\Big(D_j(x_p)\Big)$
5: **return** $D_j(x_p)$
6: **end if**
7: **else if** $\mathcal{CQ}_u.Contains(E_i.DepOnDisj.ID)$ **then**
8: Let $D_j(x_p) \leftarrow \mathcal{CQ}_u.GetNext(E_i.DepOnDisj.ID)$
9: **if** $\mathcal{CQ}_{pa}.Contains(D_c.ID) \neq$ **true**, for all $D_c \in D_j.DepOnElem.ChildDisjs$ where $D_c.ID \neq D_j.ID$ **then**
10: $\mathcal{CQ}_u.Remove\Big(D_j(x_p)\Big)$
11: **return** $D_j(x_p)$
12: **end if**
13: **else if** $\mathcal{CQ}_{pn}.Contains(E_i.DepOnDisj.ID)$ **then**
14: Let $D_j(x_p) \leftarrow \mathcal{CQ}_{pn}.GetNext(E_i.DepOnDisj.ID)$
15: Let $T \leftarrow \sum_{E_d \in D_j.ChildElems} nrw(E_d)$ where $appliedConcept(E_d) \neq$ **true**
16: **if** $\Big(degMatch(D_j)+T\Big) > degMatch(D_j.ID)$ and $\Big(degMatch(D_j)+T\Big) > degMatch(D_y)$, for all $D_y \in D_j.DepOnElem.ChildDisjs$ **then**
17: $\mathcal{CQ}_{pn}.Remove\Big(D_j(x_p)\Big)$
18: **return** $D_j(x_p)$
19: **end if**
20: **end if**
21: **return null**

where all applied elements have clashed. If $\mathcal{CQ}_{pa}$ contains any assertion $D_j(x_p)$ about an equivalent copy of D_z, then this is obtained and removed from the queue. However, this only occurs if the next disjunct element of D_j which has not yet been applied / evaluated, has a weight which is less than or equal to the disjunct element E_i passed to this algorithm. As discussed in Section 5.6.2, disjunctions D_j are stored in $\mathcal{CQ}_{pa}$ in descending relative weight $rw(E_i)$ order since a lower weight indicates fewer remaining disjunct elements which have not been applied or generated a clash.

If $\mathcal{CQ}_{pa}$ did not contain an assertion $D_j(x_p)$, then the queue $\mathcal{CQ}_u$ is checked which contains disjunctions where no elements have been applied. If $\mathcal{CQ}_u$ contains an assertion $D_j(x_p)$ where D_j is an equivalent copy of D_z, it is obtained

and returned. However, in the case that D_j depends on a parent disjunct element D_p which contains multiple child disjunctions, and an alternative child D_c has already been applied and is contained in $\mathcal{CQ}_{pa}$, then this must be given preference over D_j. For example, assume the algorithm is attempting to retrieve an assertion about the disjunction D_1. Assume that D_1 is a conjunct member of a conjunction E_1 such that $E_1 \equiv (D_1 \sqcap D_2)$ and E_1 is a weighted disjunct element. In the case that an assertion about D_2 is already contained in $\mathcal{CQ}_{pa}$, the disjunction D_1 is not applied. This is because all evaluated disjunct elements of D_2 have clashed and if the remaining elements clash, then E_1 is proven to clash meaning that D_1 does not need to be evaluated.

If $\mathcal{CQ}_u$ did not contain an assertion $D_j(x_p)$, then the queue $\mathcal{CQ}_{pn}$ is checked, which contains disjunctions where not all applied elements have clashed. If $\mathcal{CQ}_{pn}$ contains an assertion $D_j(x_p)$ about an equivalent copy the disjunction D_z then it is obtained and returned. However, it is possible that evaluating the remaining un-applied elements of D_j will not increase the current degree of match (even if evaluating these generates a clash). This is the case when the degree of match for an equivalent copy of D_z is higher than the sum of the degree of match for the D_j in the retrieved assertion $D_j(x_p)$ and the normalised relative weight $nrw(E_d)$ for all E_d where E_d is an element of D_j which has not yet been applied / evaluated. It is the same case for alternative child disjunctions D_y where $D_y \in D_j.DepOnElem.ChildDisjs$. In this situation a disjunction D_j is discarded / not applied. There is no point in applying a disjunction if this cannot increase the degree of match for the current match check. Note $appliedConcept(E_d)$ is set by Algorithm 5.9, Section 5.6.1 and $degMatch(D_j)$ is set by Algorithm 5.21 in section 5.8.

If an assertion $D_j(x_p)$ about a weighted disjunction was not found by Algorithm 5.14, then the $GetNextDisj$ algorithm, will iterate $DisjElemIter$, to the next disjunct in the iterator a weighted disjunction is found or all disjunctions have been applied.

This completes our discussion of how our adaptive inference strategy selects the next disjunction to the ⊔-rule to, based on disjunction weight. However, because the ⊔-rule generates new expansions in the expansion tree, the effect of applying the ⊔-rule in disjunction weight order is that there may be multiple simultaneous expansion branches which are open / unfinished at one time. The ⊔-rule may further expand any of the open branches, thus jumping between these. This requires a new approach to branch identifiers and management of the mTableaux ABox state. This will be discussed in the next section.

5.7 Adaptive Tableaux State Management

As we have discussed in this chapter, our adaptive inference strategy associates weights with conjunctive concepts which form conditions in a user request, and proves or disproves an inferred match by negating the conjunction to become a disjunction. In the previous section we detailed the way in which our strategy selects the next disjunction to apply the ⊔-rule to based on these weights. Since the ⊔-rule gives rise to branch expansion in the expansion tree, this expansion occurs in weight order, meaning that there may be several unfinished branches open at one time. Under our strategy, the reasoner will jump between these branches until they are all finished (or until constraints such as available time and resources are no longer met). Conversely, as was discussed in Section 3.5 standard Tableaux employs depth-first expansion. When a standard Tableaux reasoner changes from a branch node b_{i+1} to an earlier branch b_i, to continue expansion of an alternate branch, any branches and assertions which occurred after b_i are discarded. Since our adaptive strategy is not depth-first, several modifications to the branch expansion scheme are required. For instance, assume that the weight ordered application of disjunctions, gives rise to the expansion shown in Figure 5.9, where circles are branch nodes and the number contained in the circles indicates the order of branch expansion.

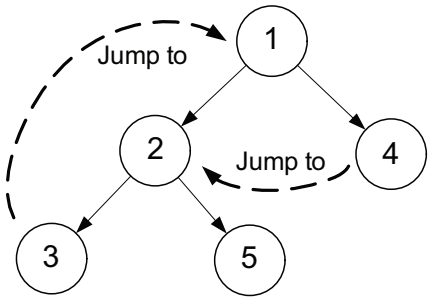


Figure 5.9: Adaptive Inference Branching Example

In this figure, there is an initial branch expansion comprising nodes 1, 2 and 3. Assume that branch node 3 generates a clash. Then branch node 4 is expanded from node 1 and assume that branch node 4 generates a clash. Finally, branch node 5 is created as an expansion from branch node 2. In standard Tableaux, a jump from branch node 3 to branch node 1 (in order to expand branch node 4), would discard all of the state changes which occurred after branch node 1, before expanding branch node 4. However, the jump from branch node 4 back to branch node 2 (in order to expand branch node 5), would result in the need to re-apply all transformations which occurred between branch 1 and 2, since these were discarded. Therefore, in order to cater for weight directed expansion, without requiring re-evaluation of transformation rules, in this section we propose a novel approach for state management in the reasoner. Under our approach, the state of several different unfinished branches are simultaneously maintained, such that the reasoner can jump between branch states. This replaces the standard Tableaux reasoner restore operations which are not performed by our adaptive strategy (see Algorithm 5.2 in Section 5.3.1).

Catering for multiple unfinished branches, requires the modification of the following components of standard Tableaux:

- Branch point node identifiers, which we will discuss in section 5.7.1;

- Sets used to maintain ABox state, including type and role labels and the $ToDo$ list of assertions which transformation rules are applicable to, which we will discuss in section 5.7.2.

5.7.1 Branch Point Node Identifiers

Standard Tableaux uses branch identifiers to identify which branch node a particular assertion depends on. This is used to determine which branch to restore the reasoner state to, when a back jump occurs, as shown in Section 3.5. Since standard Tableaux is depth-first, it identifies branches using a number / counter which is increased every time a new branch is created (i.e. a depth count). However, in our proposed adaptive strategy, there may be simultaneously open branches which the reasoner needs to jump between, in order to complete the expansion in weight directed order. Therefore, a depth counter is not sufficient to uniquely identify branch nodes in our proposed strategy.

In our adaptive inference strategy, we establish an adaptive branch identifier to replace the standard Tableaux depth count, where an adaptive branch node refers to a branch node created by application of a weighted disjunction. Recall from Section 5.3.1 that non-weighted disjunctions are applied in the standard Tableaux way, after no more weighted disjunctions can be applied for a particular expansion. Therefore, our modified branch identifiers relate only to adaptive branch nodes. Let $adapBranchID(b_k)$ denote an adaptive branch identifier for a branch node b_k. The structure of a adaptive branch identifier is shown in Equation 5.9. We use the notation "-" to represent a hyphen in our branch identifiers e.g. 3-0-1 and these are illustrated below. The square braces indicate an optional part of the adaptive branch identifier.

$$adapBranchID(b_k) = depth(b_k) \text{ "-" } breadth(b_k)\Big[\text{ "-" } copyNumber(b_k)\Big] \tag{5.9}$$

Let $depth(b_k)$ denote a depth count for a branch node b_k in the expansion tree $\mathcal{G}$, which is the same as standard Tableaux branch node identifiers. However, in addition our adaptive identifiers introduce a breadth count and optional copy number in order to maintain uniqueness. Let $breadth(b_k)$ denote the breadth count for a branch node b_k. The $breadth(b_k)$ increases for every new branch node b_k which has the same $depth(b_k)$. Furthermore, recall from Section 5.4 that application of the $\forall$-rule may result in several copies of a weighted disjunction being added as types to several different individuals. When the $\sqcup$-rule is applied to each of these disjunctions, this will create a separate branch node for each. The $copyNumber(b_k)$ is an optional additional identifier used to uniquely identify each of these separate branch nodes, which are generated by applying separate copies of the same disjunction.

Our adaptive strategy determines adaptive branch identifiers by first associating a depth $depth(E_i)$ and breadth $breadth(E_i)$ identifier to each disjunct element member E_i of each of the weighted disjunctions D_j and child disjunctions D_j which are derived from the user request. These are used to establish an initial adaptive branch identifier associated with each disjunct E_i. Only weighted disjunct elements are given identifiers (i.e. not non-weighted disjunctions). In the case that an element is copied and added as a type to multiple different individuals by the $\forall$-rule, then this rule will generate the additional $copyNumber(E_i)$ and associate it with the disjunct element E_i, and append this to the branch identifier for E_i. The full branch identifier associated with E_i will then be transferred to the branch node b_k when this is created by application of the $\sqcup$-rule generates a b_k for the element E_i. If a weighted disjunct element E_i has not been copied as a result of a $\forall$-rule and it does not depend on a weighted disjunct E_j which has been copied as a result of a $\forall$-rule, then it does not have a $copyNumber(E_i)$, since there is only one equivalent copy of E_i.

It is noteworthy, that the $depth(E_i)$ values of 0 and 1 are reserved. All explicit assertions which are loaded into the reasoner from an ontology are set to depend on an adaptive branch node dependency identifier of 0-0, such that $adapBranchID\Big(depOnAdapBranch(C_y)\Big)$ = 0-0, where C_y is added to the type label for some individual x_r. Then any assertions which are added due any pre-processing performed by the reasoner such as immediate application of $\forall$-rules and $\sqcap$-rules or other pre-processing related to the standard Tableaux optimisation strategies (Tsarkov et al., 2007) are given an adaptive branch node dependency identifier of 1-0, such that $adapBranchID\Big(depOnAdapBranch(C_y)\Big)$ = 1-0, where C_y is added to the type label for some individual x_r. When our adaptive inference strategy performs an inference check, which results in the negation of the user request definition C_z being added as a type to the service description individual x_p, this assertion depends on the adaptive branch identifier 2-0, such that $adapBranchID\Big(depOnAdapBranch(C_z)\Big)$ = 2-0.

An example of branch identifiers is as follows. Assume, a negated user request becomes the disjunction $\neg$Request $\equiv$ ($\neg$RetailOutlet $\sqcup \forall$ sells.$\neg$InternetRequest) and $\neg$InternetRequest $\equiv$ ($\neg$WiFi $\sqcup$ $\neg$Internet). This will result in the initial disjunct element identifiers such that $adapBranchID($ $\neg$Request $)$=2-0 where $\neg$Request is a $WeightedDisjunct$, $adapBranchID($ $\neg$RetailOutlet $)$=3-0, $adapBranchID($ $\neg$InternetRequest $)$=3-1, $adapBranchID($ $\neg$WiFi $)$=4-0, $adapBranchID($ $\neg$Internet $)$=4-1. However, assume that $\forall$ sells.$\neg$InternetRequest is added as a type to an individual JimsInternet which connects to inet and coffee using the role sells. Application of the $\forall$-rule will add a copy of $\neg$InternetRequest to both inet and coffee. When this occurs each copy of $\neg$InternetRequest is given a copy number starting at zero, so that the disjunct element copy added to inet is given a $copyNumber$ of 0 and the disjunct element added to coffee is given a $copyNumber$ of 1. This results in the final identifiers such that $\neg$InternetRequest added to inet has a $adapBranchID$ of 3-1-0 and $\neg$InternetRequest added to coffee has a $adapBranchID$ of 3-1-1. Since there

will be two copies of ¬InternetRequest, this means there will be two copies of ¬WiFi and ¬Internet, which depend on each ¬InternetRequest copy (since they are elements of ¬InternetRequest). Therefore, the final identifiers for the ¬WiFi element added to inet will be 4-0-0 and the copy added to coffee will be 4-0-1. The final identifiers for ¬Internet element added to inet will be 4-1-0 and the copy added to coffee will be 4-1-1.

In addition to a unique identifier, each weighted disjunct element, and thus adaptive branch node, will also have a set of identifiers which it depends upon, as we mentioned in Section 5.3.1. Let $adapDepBranchIDs(E_i)$ denote the set of adaptive branch identifies $adapBranchID(E_i)$ which a disjunct element E_i depends on. Dependant identifiers for E_i will include the branch identifier for disjunct E_i itself, as well as the identifiers of all parent disjunct elements which E_i depends on, as defined in Equation 5.10. Identifiers in the dependency set, are in ascending order.

$$\begin{aligned} adapDepBranchIDs(E_i) \;=\; & adapBranchID(E_i) \cup adapDepBranchIDs(E_j) \\ & \text{for all } E_j, \text{ where } E_j = E_i.DepOnDisj.DepOnElem, E_j \neq null, \text{ and} \\ & ID_y \leq ID_z, \text{ for all } ID_y, ID_z \in adapDepBranchIDs(E_i) \end{aligned} \tag{5.10}$$

For instance, in the example we just described, the disjunct element ¬WiFi depends on ¬InternetRequest which depends on ¬Request. As mentioned previously, there are two equivalent copies of ¬InternetRequest and its dependant element ¬WiFi. One copy of these elements will be added to the individual inet, therefore, the branch identifier dependency for ¬WiFi can be defined such that $adapDepBranchIDs(\neg\text{WiFi}) = \{0\text{-}0, 1\text{-}0, 2\text{-}0, 3\text{-}1\text{-}0, 4\text{-}0\text{-}0\}$, where ¬WiFi $\in \mathcal{L}$(inet).

We will now provide the algorithm which allocates the initial depth and breadth counts and dependencies to each disjunct element of weighted disjunctions arising from the user request, in Algorithm 5.15. We will then detail the allocation of $copyNumber(E_i)$ by application of the ∀-rule, followed by transferring branch identifiers from weighted disjunctions to new branches, as well as generating dependencies.

Algorithm 5.15 EstablishAdapBranchIDs(E_i)

Inputs: WeightedDisjunct E_i

1: **if** $breadthCount\Big(depth(E_i)\Big) =$ **null** *not yet set for $depth(E_i)$*\ **then**
2: Let $breadthCount \leftarrow 0$
3: **else**
4: $breadthCount\Big(depth(E_i)\Big) \leftarrow breadthCount\Big(depth(E_i)\Big) + 1$
5: **end if**
6: $breadth(E_i) \leftarrow breadthCount\Big(depth(E_i)\Big)$
7: Let $E_p \leftarrow E_i.DepOnDisj.DepOnElem$
8: **if** $E_p =$ **null** *first disjunct element*\ **then**
9: $depth(E_i) \leftarrow 2$
10: Let $adapBranchID(E_i) \leftarrow depth(E_i)$ + "-" + $breadth(E_i)$
11: **else**
12: $depth(E_i) \leftarrow depth(E_p) + 1$
13: Let $adapBranchID(E_i) \leftarrow depth(E_i)$ + "-" + $breadth(E_i)$
14: **end if**
15: **for all** $D_u \in E_i.ChildDisjs$ **do**
16: **for all** $E_j \in D_u.ChildElems$ **do**
17: $EstablishAdapBranchIDs(E_j)$
18: **end for**
19: **end for**

The disjunction derived from the negated user request definition is passed to Algorithm 5.15 as input E_i. The notation for weighted disjunction and disjunction properties was detailed in Section 5.4. Let $breadthCount(d)$ denote a property which maintains a global breadth count for each depth level d. If the $breadthCount(d)$ has not yet been set for a depth d, then it is initialised to zero, otherwise it is incremented. The breadth of the element E_i is set to the breadth count for the current depth level. If E_i is the first disjunct element, meaning it does not depend on any parent disjunct elements, then it is initialised to a depth of 2, since 0 and 1 are reserved as mentioned earlier in this section. Alternatively, if E_i is not the first element, and it does depend on

parent elements, then the depth is incremented. The initial adaptive identifier for E_i is then set (which does not yet include a $copyNumber(E_i)$). Finally, the algorithm loops all child disjunctions D_u which depend on E_i, and then all disjunct elements E_j of each D_u, and recursively passes E_j as the input to $EstalishAdapBranchIDs$, so that each child element can be assigned with an initial branch identifier.

Now we will discuss the allocation of $copyNumber(E_i)$ which is assigned when a disjunct element E_i is copied by the $\forall$-rule. Algorithm 5.16 updates the $copyNumber(E_i)$ for a weighted disjunct element E_i. It is called whenever a duplicate copy is made a disjunct element E_i. This occurs when the $\forall$-rule adds E_i as a type to multiple individuals, or if E_i depends on a disjunct element which have been added as a type to multiple individuals. When called, Algorithm 5.16 is provided with a copy index which is incremented each time another copy of E_i is generated and added as a type to an individual. This copy index $copyNumber(E_i)$ is then appended to the adaptive branch identifier.

Algorithm 5.16 UpdateCopyNbr(E_i, $copyIndex$)

Inputs: WeightedDisjunct E_i, Integer $copyIndex$

1: $copyNumber(E_i) \leftarrow copyIndex$
2: $adapBranchID(E_i) \leftarrow depth(E_i)$ + "-" + $breadth(E_i)$ + "-" + $copyNumber(E_i)$

Now we will discuss the transferal of identifiers from disjunct elements to adaptive branch nodes. Each weighted disjunct element E_i of each weighted disjunction D_u, represents an alternative expansion in the tree, which must be explored. Therefore, in our adaptive strategy, when the $\sqcup$-rule is applied to a disjunction D_u, it generates a new adaptive branch node b_k in order to explore the expansion E_i. Therefore, Algorithm 5.17 is called by the $\sqcup$-rule (which was defined in Algorithm 5.9 in Section 5.6.1) and transfers the adaptive branch identifier $adapBranchID(E_i)$ from E_i to the new branch new b_k. It also establishes the set of adaptive branch identifiers $adapDepBranchIDs(E_i)$ which E_i, and thus b_k, depend on. The algorithm receives as input the newly

created adaptive branch node b_k and the weighted disjunct element E_i which resulted in the creation of b_k.

Algorithm 5.17 AssignAdapBranchID(b_k, E_i)

Inputs: BranchNode b_k, WeightedDisjunct E_i

1: Let $E_j \leftarrow E_i.DepOnDisj.DepOnElem$
2: Let $brID \leftarrow depBranchID(E_i)$
3: $adapBranchID(b_k) \leftarrow adapBranchID(E_i)$
4: **if** $E_j =$ **null then**
5: $\quad adapDepBranchIDs(E_i) \leftarrow \{brID\}$
6: **else**
7: $\quad adapDepBranchIDs(E_i) \leftarrow adapDepBranchIDs(E_j) \cup \{brID\}$
8: **end if**
9: $adapDepBranchIDs(b_k) \leftarrow adapDepBranchIDs(E_i)$

In the algorithm, if E_i is the top most weighted disjunct element which does not have a parent disjunct element E_j, then the dependency set includes only the adaptive branch identifier for E_i. Alternatively, if E_i does depend on a parent disjunct element E_j, then the dependency set for E_i is equal to the set for E_j with the additional inclusion of its own branch identifier. Finally, the set of branch identifiers which E_i depends on, is transferred to the new branch b_k. It is noteworthy that the adaptive identifiers will always be established for disjunct elements E_j, before elements E_i which depend on E_j, because parent disjunctions are always applied before child disjunctions, in relative weight order (see Section 5.5).

In this section, we have detailed our proposed identifiers and dependencies for adaptive branch nodes in our expansion tree. These identifiers support unique identification of these nodes, even when multiple unfinished branch expansions exist in the expansion tree, which is not supported by standard Tableaux. We will use these in the next section to manage branch node directed reasoner state.

5.7.2 Labels $\mathcal{L}$ and $ToDo$ List State

As we discussed at the beginning of this section, in our adaptive reasoning strategy, there may be simultaneous unfinished branches in the reasoner. Our adaptive strategy requires a mechanism to jump between separate branch nodes, without having to re-apply the transformations which led to a particular state at a particular branch node. This requires modifications to standard Tableaux, which only has one active branch, and discards branches and assertions which occur after a branch which the reasoner is jumping back to. Our strategy maintains simultaneous states associated with each branch node, such that when the reasoner jumps between branch nodes, the state reflects the current branch node.

As we described in Section 3.5, in standard Tableaux the addition or removal of a class concept from a type label, or role from a role label, depends on the branch node which was active at the time that the addition / removal took place. The active branch node b_{k+1} is that which has the highest identifier, such that $branchID(b_{k+1}) \geq branchID(b_k)$ for all of those branch nodes b_k contained in the expansion tree $\mathcal{G}$, where the branch identifier is a depth count. This dependency is denoted by $depOnBranch(\alpha_i) \leftarrow b_k$, where α_i is some element (e.g. an assertion) which depends on b_k. Let $\mathcal{AS}$ denote a dependency indexed set which may have elements α_i added or removed and this action is indexed using $depOnBranch(\alpha_i)$. Dependency indexed sets $\mathcal{AS}$ in standard Tableaux reasoners include:

1. Type labels $\mathcal{L}(x_r)$ containing class concepts C which have been added as a type for the individual x_r and role labels $\mathcal{L}(\langle x_r, x_p \rangle)$ containing roles R connecting an individual x_r to x_p;

2. The $ToDo$ list which contains assertions $C_j(x_r)$ that Tableaux transformation rules are applicable to.

The set $\mathcal{AS}$ is modified to support our adaptive strategy as follows. Additions or removals of elements to a dependency indexed set $\mathcal{AS}$, will be indexed by the adaptive branch node b_k in the expansion tree $\mathcal{G}$, which they depend on. An adaptive branch node is a branch node which was created as a result of the application of a weighted disjunction. Let $adapDepOnBranch(\alpha_i)$ denote the adaptive branch which an element α_i depends on. Let $\mathcal{G}.activeAdapBranch$ denote the currently / last active branch adaptive identifier for the expansion tree $\mathcal{G}$. This is set by the $\sqcup$-rule, when it is applied to a weighted disjunction resulting in the creation of a new branch node as defined in Algorithm 5.9 in Section 5.6.1.

In our proposed adaptive strategy, if a particular class concept generates a clash, and this concept depends on an adaptive branch node, then the reasoner state will not be changed to an earlier state using a reasoner restore which discards assertions as in standard Tableaux. Rather, our strategy will retain the current branch and those actions (e.g. assertions) which depend on it, when jumping to an alternative branch node.

Additionally, as we discussed in Section 5.6, non-weighted disjunctions can still be applied, but only after no further weighted disjunctions can be applied for a particular branch expansion in $\mathcal{G}$. In the situation that a non-weighted disjunction is applied, this occurs in the same way as in standard Tableaux, thus creating standard Tableaux branch point nodes b_{k_1} which extend from adaptive branch point nodes b_k which exist earlier in the expansion tree $\mathcal{G}$. Elements (e.g. assertions) α_i which are added / removed after a non-weighted disjunction has been applied, will depend on a standard branch using the standard dependency $depOnBranch(\alpha_i) = b_{k+1}$, where α_i depends on b_{k+1}, as defined in Algorithm 3.2 in Section 3.5.1. However, in addition to this, these elements α_i will also depend on the last active adaptive branch node $\mathcal{G}.activeAdapBranch$, such that $depOnAdapBranch(\alpha_i) = \mathcal{G}.activeAdapBranch$.

Having now established that every element added / removed from the set $\mathcal{AS}$ will be indexed by adaptive branch node (in addition to standard branch node if applicable), we will now describe the way in which the set $\mathcal{AS}$ is modified to support multiple state views. The contents of an instance of $\mathcal{AS}$ contains only those elements which reflect the state, given a last active adaptive branch node $\mathcal{G}.activeAdapBranch$. We will use a seperate instance of the set $\mathcal{AS}$ to maintain the state of:

1. Each type label $\mathcal{L}(x_r)$ which contains class concepts C which have been added as a type for the individual x_r and each role label $\mathcal{L}(\langle x_r, x_p \rangle)$ which contains roles R connecting an individual x_r to x_p;

2. The $\mathcal{DQ}$ queue (which was defined in Section 5.6.2) that contains non-weighted disjunctions to which the $\sqcup$-rule is applicable to. It is noteworthy that if the mTableaux caching strategy (CS) (which was defined in Section 4.5) is enabled, then $\mathcal{DQ}$ will be replaced by the CS $\mathcal{Q}^d$. In this case, $\mathcal{Q}^d$ will be represented using $\mathcal{AS}$;

3. The standard Tableaux $ToDo$ list which contains all assertions for which transformation rules are applicable to, excluding assertions about disjunctions. Weighted disjunctions are stored in $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$ or $\mathcal{CQ}_{pn}$ and non-weighted disjunctions are contained in $\mathcal{DQ}$ as described in Section 5.6.2. It is noteworthy that, if mTableaux caching strategy (CS) which was defined in Section 4.5, is enabled then the weight ordered CS queue $\mathcal{Q}^o$ is used instead of the $ToDo$ list, to maintain all assertions for which transformation rules are applicable to, excluding assertions about disjunctions. In this case, $\mathcal{Q}^o$ will be represented using $\mathcal{AS}$. Also note, $\mathcal{CQ}_u$, $\mathcal{CQ}_{pa}$ and $\mathcal{CQ}_{pn}$ do not need to be stored in a dependency indexed set $\mathcal{AS}$ because these contain weighted disjunctions which guide the expansion process and relative weights ensure that disjunctions are applied in an order which maintains integrity (e.g. relative weights ensure that

a parent disjunction is applied before a child disjunction, as defined in Section 5.5);

4. If our ST mTableaux optimisation strategy, outlined in Section 4.3, is enabled, then the set $\mathcal{ST}$ of individuals which transformation rules are applicable to, will need to be represented using $\mathcal{AS}$.

Now we will define the multi-state functionality of $\mathcal{AS}$ in more detail. The main approach, is that we will obtain the set of dependencies for the currently active adaptive branch node $\mathcal{G}.activeAdapBranch$. As defined in the previous section, given an adaptive branch node b_k, the adaptive identifier for this node is given by $adapBranchID(b_k)$. Additionally, the set of identifiers which b_k depends on (including itself), is given by $adapDepBranchIDs(b_k)$. The contents of a set $\mathcal{AS}$ is determined by the currently active adaptive branch node. Moreover, the current state view of $\mathcal{AS}$, contains only those elements α_i which depend on an adaptive branch node identifier, which is contained in the set of adaptive branch node identifiers for the active adaptive branch node $adapDepBranchIDs(\mathcal{G}.activeAdapBranch)$.

In order to manage this process, we need to record all additions and removals from the set $\mathcal{AS}$. Let $\mathcal{AS}.added$ contain all elements α_i which have been added to the set $\mathcal{AS}$. Let $\mathcal{AS}.removed$ contain all elements α_i which have been removed from the set $\mathcal{AS}$. All additions and removals α_i are indexed by $depOnAdapBranch(\alpha_i)$. When an assertion is added or removed it depends on the currently active branch at the time that it was added or removed, such that $depOnAdapBranch(\alpha_i) \leftarrow \mathcal{G}.activeAdapBranch$. The contents of $\mathcal{AS}$ is returned by Algorithm 5.18, which is given the set $\mathcal{AS}$, and the active branch b_k as input, where $b_k \leftarrow \mathcal{G}.activeAdapBranch$.

The algorithm returns a set V of elements α_i, which represent the state view of $\mathcal{AS}$, influenced by the active branch node. If V can be obtained from the cache, then a cached V is returned (the cache will be defined in Algorithm 5.19 later in this section). If V is not in the cache then the returned set is

Algorithm 5.18 GetState($\mathcal{AS}$, b_k)

Inputs: AdaptiveSet $\mathcal{AS}$, BranchNode b_k
Pre-conditions: $b_k = \mathcal{G}.activeAdapBranch$
Outputs: Set V

1: **if** $getFromCache(\mathcal{AS}, b_k) \neq$ **null then**
2: **return** $getFromCache(\mathcal{AS}, b_k)$
3: **end if**
4: Let $V \leftarrow \emptyset$
5: **for all** $adapID_y \in adapDepBranchIDs(b_k)$, such that $adapID_y \leq adapID_z$, $1 \leq y \leq z \leq m$ **do**
6: **for all** $\alpha_i \in \mathcal{AS}.added$, where $adapBranchID\Big(depOnAdapBranchNode(\alpha_i)\Big) = adapID_y$ **do**
7: $V \leftarrow V \cup \{\alpha_i\}$
8: **end for**
9: **for all** $\alpha_i \in \mathcal{AS}.removed$ where $adapBranchID\Big(depOnAdapBranchNode(\alpha_i)\Big) = adapID_y$ **do**
10: remove α_i from V
11: **end for**
12: **end for**
13: $\mathcal{AS}.cache \leftarrow V$
14: $\mathcal{AS}.lastAdapBranch \leftarrow b_k$
15: **return** V

constructed by iterating through the set of branch identifiers which b_k depends $adapDepBranchIDs(b_k)$, in ascending order. For each $adapID_y \in adapDep$-$BranchIDs(b_k)$, any elements α_i in $\mathcal{AS}.added$ which depend on $adapID_y$ are added to V, then any elements α_i in $\mathcal{AS}.removed$ which depend on $adapID_y$ are then removed from V. Once V has been constructed it is added to the cache. Let $\mathcal{AS}.cache$ denote an attribute containing the cache for $\mathcal{AS}$. Let $\mathcal{AS}.lastAdapBranch$ denote the branch node b_k which the cache depends on. The main purpose of the cache, is to avoid repeatedly reconstructing the set $\mathcal{AS}$ when successive child branch nodes in $\mathcal{G}$ of the same branch are being visited. Using the cache means that $\mathcal{AS}$ only needs to be reconstructed when the adaptive inference strategy jumps between branches (i.e. a jump from branch node b_o to b_k where b_o is dependent on at least one different branch node than b_k). Algorithm 5.19 defines the process which attempts to retrieve a set from the cache.

Algorithm 5.19 GetFromCache($\mathcal{AS}$, b_k)

Inputs: AdaptiveSet $\mathcal{AS}$, BranchNode b_k
Pre-conditions: $b_k = \mathcal{G}.activeAdapBranch$
Outputs: Set V

```
1: if AS.cache = null then
2:    return null
3: end if
4: Let depth_curr be the depth value of adapBranchID(b_k)
5: Let depth_cached be the depth value of AS.lastAdapBranch
6: Let depIDS_curr ← adapDepBranchIDs(b_k)
7: Let depIDS_cached ← adapDepBranchIDs(AS.lastAdapBranch)
8: if depth_curr ≥ depth_cached and depIDS_cached ⊆ depIDS_curr then
9:    AS.lastAdapBranch ← b_k \*update to current branch*\
10: else
11:    AS.cache ← null
12: end if
13: return AS.cache
```

A set $\mathcal{AS}$ and the currently active branch b_k are passed as inputs to the algorithm, which returns the last cache entry $\mathcal{AS}.cache$. However, this cached entry is only returned if b_k has a depth value which is the same or equal the depth of the branch node which the cache depends on $\mathcal{AS}.lastAdapBranch$. The depth value of an adaptive branch node, is the first number in the identifier which occurs before the first hyphen (e.g. the depth value of 1-0-0 is 1). Additionally, set of adaptive branch identifiers which the cached adaptive branch node $\mathcal{AS}.lastAdapBranch$ depends on must be a subset of those which the current branch b_k depends on. If these conditions are not met, this implies that a branch jump (rather than the continuation of a branch) has occurred, and the cache entry is removed.

Algorithm 5.20 illustrates the procedure involved in adding a new element α_i to a set $\mathcal{AS}$, where b_k is the active adaptive branch node.

The element α_i being added to $\mathcal{AS}$ is set to depend on b_k. The element α_i is added to $\mathcal{AS}.added$. It is also added to the cache if there is cache entry for $\mathcal{AS}$ which is valid for the current b_k. The procedure is the same when removing an element α_i except that line 3 of Algorithm 5.20 is replaced with remove α_i from $\mathcal{AS}.cache$ and line 6 is replaced with $\mathcal{AS}.removed \leftarrow \mathcal{AS}.removed \cup \{\alpha_i\}$.

Algorithm 5.20 Add($\mathcal{AS}$, α_i)

Inputs: Set $\mathcal{AS}$, Element α_i, BranchNode b_k
Pre-conditions: $b_k = \mathcal{G}.activeAdapBranch$
Outputs: Set V

1: $depOnAdapBranch(\alpha_i) \leftarrow b_k$
2: **if** $getFromCache(\mathcal{AS}, b_k) \neq$ **null then**
3: $\quad \mathcal{AS}.cache \leftarrow \mathcal{AS}.cache \cup \{\alpha_i\}$
4: $\quad \mathcal{AS}.lastAdapBranch \leftarrow b_k$
5: **end if**
6: $\mathcal{AS}.added \leftarrow \mathcal{AS}.added \cup \{\alpha_i\}$

Now we illustrate an example state view V of $\mathcal{AS}$ at branch node b_h. In this example assume that $\mathcal{AS}_u$ is being used to represent the class concepts C_i which have been added / moved from the type label $\mathcal{L}(x_r)$ for an individual x_r, at the branch node b_h as illustrated in Table 5.3.

Add/remove concept C_i from $\mathcal{AS}_u$	$b_h = depOnAdapBranch(C_i)$
$\mathcal{AS}.added \leftarrow \mathcal{AS}.added \cup \{C_1\}$	2-1
$\mathcal{AS}.added \leftarrow \mathcal{AS}.added \cup \{C_2\}$	2-3
$\mathcal{AS}.removed \leftarrow \mathcal{AS}.removed \cup \{C_1\}$	3-1-0
$\mathcal{AS}.added \leftarrow \mathcal{AS}.added \cup \{C_3\}$	3-1-0
$\mathcal{AS}.added \leftarrow \mathcal{AS}.added \cup \{C_4\}$	4-0-0

Table 5.3: Example of Adaptive Set $\mathcal{AS}$ State Management

The currently active adaptive branch node $\mathcal{G}.activeAdapBranch$ has the identifier of 4-0-0. Assume that the currently active adaptive branch node depends on the branches with the identifiers such that $depOnAdapBranch - IDs(\mathcal{G}.activeAdapBranch) = \{2\text{-}1,\ 3\text{-}1\text{-}0,\ 4\text{-}0\text{-}0\}$. This will result in a state view $GetState(\mathcal{AS}_u, \mathcal{G}.activeAdapBranch)$ of $\{C_3, C_4\}$. The actions which are valid for the current state view (i.e. at adaptive branch identifier 4-0-0) are highlighted in yellow in the table. The class concept C_1 is not present in the state view because although it was added at adaptive branch node 2-1, it was later removed at adaptive branch node 3-1-0. The class concept C_2 was excluded from the set, because it was added at branch identifier 2-3, which is not contained in the list of identifiers which the active adaptive branch node depends on.

This concludes our discussion on reasoner state management. In this chapter, we have now discussed the way in which we establish weights and use these to control the order of branch expansion. We have also discussed the way in which we maintain state in order to support jumping between multiple unfinished branches without having re-apply transformation rules, as would be required using standard Tableaux. Our proposed approach enables priority based "anytime" incremental matching which can be interrupted at any stage during the matching process based on to constraints such as time or resources. In the next section we will outline our weighted degree of match metric, which can be used to provide a result to the user based on the evaluations / computations actually completed at the time of interruption.

5.8 Degree of Match

In this section, we define our metric which provides a weighted degree of match value of service description to the user request. As stated in Section 5.1, our adaptive inference strategy supports incremental "anytime" matching, which can be interrupted prematurely based on user constraints such as constrained time or resources. Therefore, our degree of match is updated incrementally, every time a condition in the service request is found to match a feature of the service description. As such, a degree of match result can be provided based on the processing / computations performed up to the point of the interruption. Additionally, a degree of match ensures that our strategy supports partial matching, in the case that the service description does not completely match the user request. These notions of partial service matching have been widely employed in matching literature such as Skoutas et al. (2007); Lu (2005) and Srinivasan et al. (2005). However, partial matching is not supported by current semantic reasoners which provide only a binary $true$ or $false$ result. A $true$ result is provided only if the service is found to completely match the

user request. In all other cases a $false$ result is provided and if reasoning is interrupted early, no result is provided at all.

Now we focus our discussion on how our adaptive inference strategy generates a degree of match. We maintain a separate degree of match value for each weighted disjunct element E_i, of every weighted disjunction D_j, and weighted disjunction identifier $D_j.ID$ for equivalent disjunction copies. Let $degMatch(B)$ denote the current degree of match for the disjunct, disjunction or identifier B. The final degree of match for the user request to the service description is given by $degMatch(\neg E_0)$ where $\neg E_0$ is the negated user request represented as a weighted disjunct element. The degree of match value for $\neg E_0$ is calculated as the weighted sum of those request conditions which matched the service description and is a value between 0 and 1, where 1 indicates a full match and 0 indicates no match. More specifically, it is the sum of the normalised relative weight $nrw(E_i)$ values for all leaf weighted disjunct elements E_i (i.e. those which do not contain any child disjunctions) which matched the service description.

However, as we discussed in Section 5.4 if multiple weighted disjunctions or equivalent copies of the same disjunction depend on the same weighted disjunct element E_p, then only one of these disjunctions needs to clash for all expansions in order to generate a clash for E_p. Therefore, where multiple weighted disjunctions D_j depend on the same weighted disjunct E_p, only the disjunction D_j with the highest match value should be used when calculating the degree of match for E_p. As outlined in Section 5.4, there are two cases where multiple weighted disjunctions D_j may depend on the same weighted disjunct E_p:

1. in the case that a weighted disjunct element E_p is a conjunction of multiple disjunctions $E_p \equiv D_j \sqcap D_{j+1}$, these both depend on the conjunction

E_p. If any of these disjunctions D_j clash for all expansions, then the conjunction E_p is proven to clash, and the other disjunctions do not need to be checked;

2. in the case that a weighted disjunct element E_p is a universal quantifier $E_p \equiv \forall R.D_j$, the disjunction D_j may be copied and added as types to several different individuals. If any of these disjunctions D_j clash for all expansions, then the universal quantifier E_p is proven to clash, and the other disjunction copies do not need to be checked.

Therefore, we define degree of match as the is the sum of $nrw(E_i)$ for all weighted disjunct elements E_i, where E_i has generated a clash andE_i has no child disjunctions, such that $clashingConcept(E_i)$ and $E_i.ChildDisjs = \emptyset$. However, due to the cases we just described where a weighted disjunct element E_i has multiple dependent disjunctions D_j, only the one disjunction D_j with the highest sum of $nrw(E_c)$ for all disjunct element E_c members of D_j, where $clashingConcept(E_) = true$, is added to the degree of match value. This functionality is captured in Algorithm 5.21, which updates the degree of match whenever a clash occurs. This algorithm is called by Algorithm 5.2 which we defined in Section 5.3.1, whenever a weighted disjunct element E_i is found to clash. The algorithm is given E_i as the first input, and the $nrw(E_i)$ value of E_i is passed as the second input parameter d, such that $d \leftarrow nrw(E)$.

The process of the algorithm is as follows. Firstly, if d represents an increase of the current degree of match for the element E_i, then both E_i and the disjunction D_j which E_i depends on (is a member of), should be increased by the difference. If the degree of match for E_i is already higher than d, this may be because E_i is a conjunction containing other child disjunctions and one of these has already received a higher degree of match. To cater for multiple equivalent copies of D_j, due to a $\forall$-rule, a separate degree of match value for $D_j.ID$ is also kept. If the degree of match for D_j is higher than the degree of match for $D_j.ID$, then D_j is the disjunction copy with the highest

Algorithm 5.21 UpdateResultsOnClash(E_i, d)

Inputs: WeightedDisjunct E_i, Double d

1: **if** $degMatch(E_i) < d$ **then**
2: Let $increase \leftarrow \Big(d - degMatch(E_i)\Big)$
3: $degMatch(E_i) \leftarrow degMatch(E_i) + increase$
4: $degMatch(E_i.DepOnDisj) \leftarrow degMatch(E_i.DepOnDisj) + increase$
5: **if** $degMatch(E_i.DepOnDisj.ID) < degMatch(E_i.DepOnDisj)$ **then**
6: $degMatch(E_i.DepOnDisj.ID) \leftarrow degMatch(E.DepOnDisj)$
7: $d \leftarrow degMatch(E_i.DepOnDisj.ID)$
8: **end if**
9: **else**
10: $d \leftarrow 0$ *no increase*\
11: **end if**
12: Let $E_p \leftarrow E_i.DepOnDisj.DepOnElem$
13: **if** $d > 0$ and $E_p \neq$ **null** and $E_p.DepOnDisj \neq$ **null** **then**
14: $UpdateResultsOnClash(E_p, d)$
15: **end if**

degree of match. Therefore, the difference is added to the degree of match for $D_j.ID$. As such $D_j.ID$ always has the highest degree of match of all equivalent copies. The variable d, which contains a double value, is then set to contain the degree of match for $D_j.ID$ and this is recursively propagated to the weighted disjunct element which D_j depends on. The increase to degree of match is thereby propagated up until either the top most disjunction is reached, which represents the negated user request itself, or there is no increase to propagate. That is, in the case that any D_j has a degree of match which is equal to or less than $D_j.ID$, then there is no increase to propagate to parent elements. The degree of match between a user request $\neg E_0$ and the service description is $degMatch(\neg E_0)$, such that $0 \leq degMatch(\neg E_0) \leq 1$ where 1 indicates a full match and $\neg E_0$ is the negated user request, which is represented as weighted disjunct element. This concludes the discussion of our adaptive inference strategy.

5.9 Summary

Current semantic reasoners require that a matching task is completed in full before any result is provided, operating on an "all or nothing" principle. In the case that the matching process is interrupted before its completion then no result is provided at all. Additionally, if any condition in the user request fails to match, then a negative match result is provided, even if this condition was not a significant requirement to the user. These characteristics provide an opportunity to introduce a more flexible approach to reasoning in a resource constrained environment where constraints, such as time and resource availability, need to be considered. Therefore, in this chapter we have proposed and developed an adaptive inference strategy. Our strategy supports priority based "anytime" incremental matching of user requirements and provides a weighted degree of match value to the user based on the request requirements checked. The user associates explicit weight values against requirements in the user request, which is represented as a conjunction. These weight values are used to generate relative weights used to prioritise the order in which conjunct elements of the request are evaluated. Since Tableaux proves or disproves an inference by negating the user request, the conjunction becomes a disjunction which generates expansion in the Tableaux expansion tree. This implies that our strategy employs priority based expansion, rather than depth-first expansion which is employed by standard Tableaux. This requires the use of queues and priorities in order to select which disjunction to expand next while adhering to the constraints implied by Tableaux transformation rules. Furthermore, weight ordered expansion means that there may be several unfinished branches in the expansion tree at one time. This requires changes to the way in which the reasoner maintains state. Our proposed strategy maintains multiple state views to enable jumping between several different branches in the expansion tree, in order to complete the matching process in priority order. Weight ordered expansion continues until the user constraints are exceeded, such as

insufficient resources or time available. Our incremental approach to calculating a degree of match means that this metric can be provided to the user at any stage during the reasoning process.

We have now completed our discussion of our light-weight and adaptive approach to enable efficient and accurate on-board mobile matching. We have implemented and evaluated our approach on both mobile and desktop environments, which we will present in the next chapter.

Chapter 6

Implementation and Evaluation

6.1 Introduction

In Chapters 4 and 5 we outlined our strategies for light-weight and adaptive inference to facilitate efficient and accurate matching of user requests with service descriptions in mobile environments. In this chapter we begin by describing the implementation of these strategies as an extension to the Pellet reasoner, in Section 6.2. We then provide two main case studies in Section 6.3 which we then use to evaluate our strategies for efficiency and accuracy. We provide two main evaluations:

1. In Section 6.4 we evaluate our light-weight mTableaux inference strategies. We first compare our strategies with current reasoners on a desktop PC, in terms of efficiency and accuracy. This is presented in Section 6.4.2. We then evaluate mTableaux on a mobile device to show that it improves computational performance efficiency to enable mobile reasoning on a resource constrained device and evaluate which strategies or combination of strategies work best together. This is presented in Section 6.4.3.

2. In Section 6.5 we evaluate our adaptive inference strategy and show that it supports incremental, priority based, reasoning and provides a degree of match result.

The results presented in this chapter have been published in (Steller et al., 2009c,a; Steller and Krishnaswamy, 2008b).

6.2 Implementation

We have implemented our light-weight mTableaux optimisation and caching strategies which were outlined in Chapter 4, and our adaptive reasoning strategy which was outlined in Chapter 5 as an extension to the Pellet 1.5[1] reasoner, as a proof of concept of our proposed approach. Pellet 1.5 supports OWL-DL with $\mathcal{SHOIQ}$ expressiveness. We used Pellet because it is an open source reasoner. Additionally, we selected Pellet over other reasoners such as FaCT++[2] because Pellet is written in Java while FaCT++ is written in C++. A Java implementation makes Pellet more portable to heterogeneous devices such as PDAs and mobile phones. In addition, unlike other reasoners, Pellet was designed from the outset to support OWL-DL based reasoning while most other reasoners were subsequently modified to support OWL-DL. Therefore, in keeping with the OWL-DL[3] specification, Pellet does not make a Unique Name Assumption (UNA), provides XML data type reasoning and it has a small core reasoning engine which is suitable for extensions (Sirin et al., 2007). Figure 6.1 shows the main components of Pellet with our extensions. The figure is adapted from Sirin et al. (2007).

Pellet incorporates components from other APIs which are shown in green in the figure. The components in blue represent Pellet implementations. The components shown in yellow represent our extensions to Pellet which implement our proposed strategies from Chapters 4 and 5.

[1] http://clarkparsia.com/pellet/ (accessed May 2009)
[2] http://owl.man.ac.uk/factplusplus/ (accessed May 2009)
[3] http://www.w3.org/TR/owl-features/ (accessed May 2009)

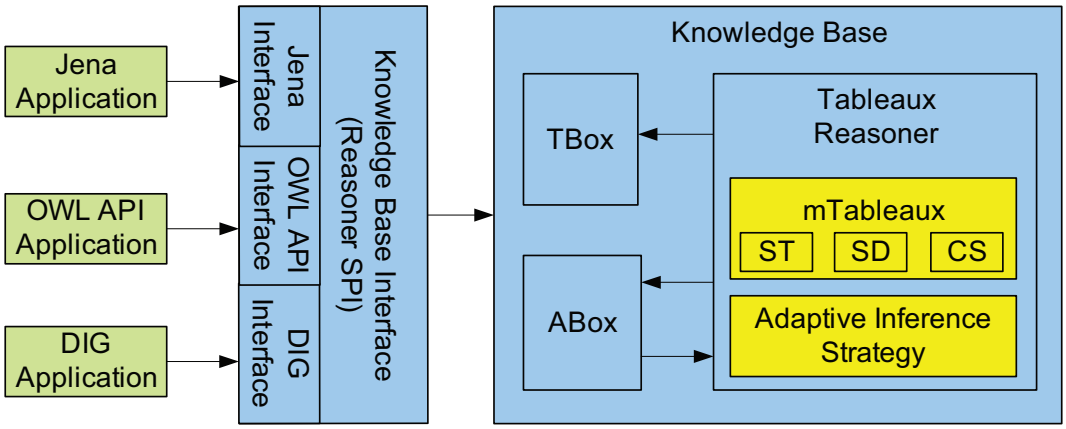


Figure 6.1: Main components of Pellet with our mTableaux and Adaptive Inference Strategy extensions

The main functionality of Pellet can be described as follows. The reasoner can be interacted with directly using an ontology API such as Jena[4] or WonderWeb OWL API[5], or remotely over HTTP using the DL Implementation Group (DIG)[6] (Bechhofer et al., 2003) interface. We interact with Pellet using Jena because it is a mature API which has undergone continuous development, growing out of the HP Labs Semantic Web Programme[7].

Initially the Jena API loads the definitions and individuals from an XML/-RDF ontology into the Pellet Knowledge Base component using the Jena Knowledge Base Service Programming Interface (SPI). Definitions are loaded into the TBox and individuals are loaded into the ABox (see Section 3.3). Various transformations are performed on the definitions in the TBox. These transformations implement the standard Tableaux optimisation strategies (Tsarkov et al., 2007) described in Section 3.4.

The Jena API can then be used to ask the reasoner whether a particular individual (service description in the ABox has a particular class concept definition (user request) as a type. If this is not found in the ABox, then the Tableaux reasoner performs the satisfiability check operation to prove or

[4]http://jena.sourceforge.net/ (accessed May 2009)
[5]http://owlapi.sourceforge.net/ (accessed May 2009)
[6]http://dl.kr.org/dig/ (accessed May 2009)
[7]http://www.hpl.hp.com/semweb/ (accessed May 2009)

disprove the inference (i.e. whether the service description individual matches the user request definition). The satisfiability check operation repeatedly applies Tableaux transformation rules to individuals in the ABox, until a clash (a contradiction) is detected for the label of an individual, or until a clash free graph is found to which no more transformations are applicable as described in Section 3.4.

We have extended the Pellet Tableaux reasoner to include our mTableaux optimisation and caching strategies which were proposed and developed in Chapter 5 (shown in yellow in Figure 6.1):

- Selective application of transformation rules (ST);
- Selective application of disjunction transformation rules (SD);
- Caching strategy (CS)

We have also extended the Tableaux reasoner to implement our adaptive inference strategy which was proposed and developed in Chapter 5. mTableaux and our adaptive inference strategy are operational on the Windows Mobile platform using the Mysaifu[8] Java J2SE Virtual Machine (JVM).

In the remainder of this chapter we provide an evaluation of our light-weight mTableaux and adaptive inference strategies which have been implemented as extensions of the Pellet reasoner. In the next section we outline two case studies which we used to evaluate our mTableaux and adaptive reasoning strategies.

6.3 Case Studies

In this section we provide two case studies which we used to evaluate our light-weight mTableaux and adaptive reasoning strategies. The mTableaux evaluation is presented in the next section. Each case study utilises separate ontologies which were used to describe the services available. These ontologies

[8]http://www2s.biglobe.ne.jp/ dat/java/project/jvm/index_en.html (accessed May 2009)

were downloaded to the user's mobile device. Later we will present request definitions which are matched against the service descriptions provided in the ontologies defined in this section.

6.3.1 Product Case Study 1: Searching for a Movie Cinema / Internet Cafe

Bob is in a foreign city centre and has walked past several stores. While moving around the city, Bob has passed an information kiosk and several shop fronts which have short range bluetooth connectivity. This bluetooth connection allows free download of ontologies containing service descriptions of products and services offered. Bob's PDA has automatically downloaded these ontologies as he passed the kiosk and the ontologies have been stored on the SD card in his PDA.

Figure 6.2 illustrates one of the service descriptions which is contained in the ontologies which has been downloaded to Bob's mobile device. In the figure, ontology individuals are depicted as ovals, role connections are depicted as directed arcs and class concept types are depicted as rectangles. An individual which has a class type is depicted as a directed connection from an individual to a class using the role owl:hasType.

The individual MovieCin1 represents a service description for a movie cinema with a cafe selling tea / coffee, Internet access and photo printing. The individual MovieCin1 is a retail outlet which sells the screening of Titanic and it has a movie cinema Cinema1. MovieCin1 also has a cafe contained within it represented by the individual MovCafe1. The MovCafe1 individual is specified as selling 30 products (not all of these are shown in the figure). In particular, MovCafe1 sells various kinds of food, has Internet access, a photo kiosk and a public phone. The Internet access has a desktop PC component (as opposed to only a WiFi hot spot) which has a CD Burner and SD card reader. In addition, the photo kiosk has an SD card reader and outputs 10x15 inch photo

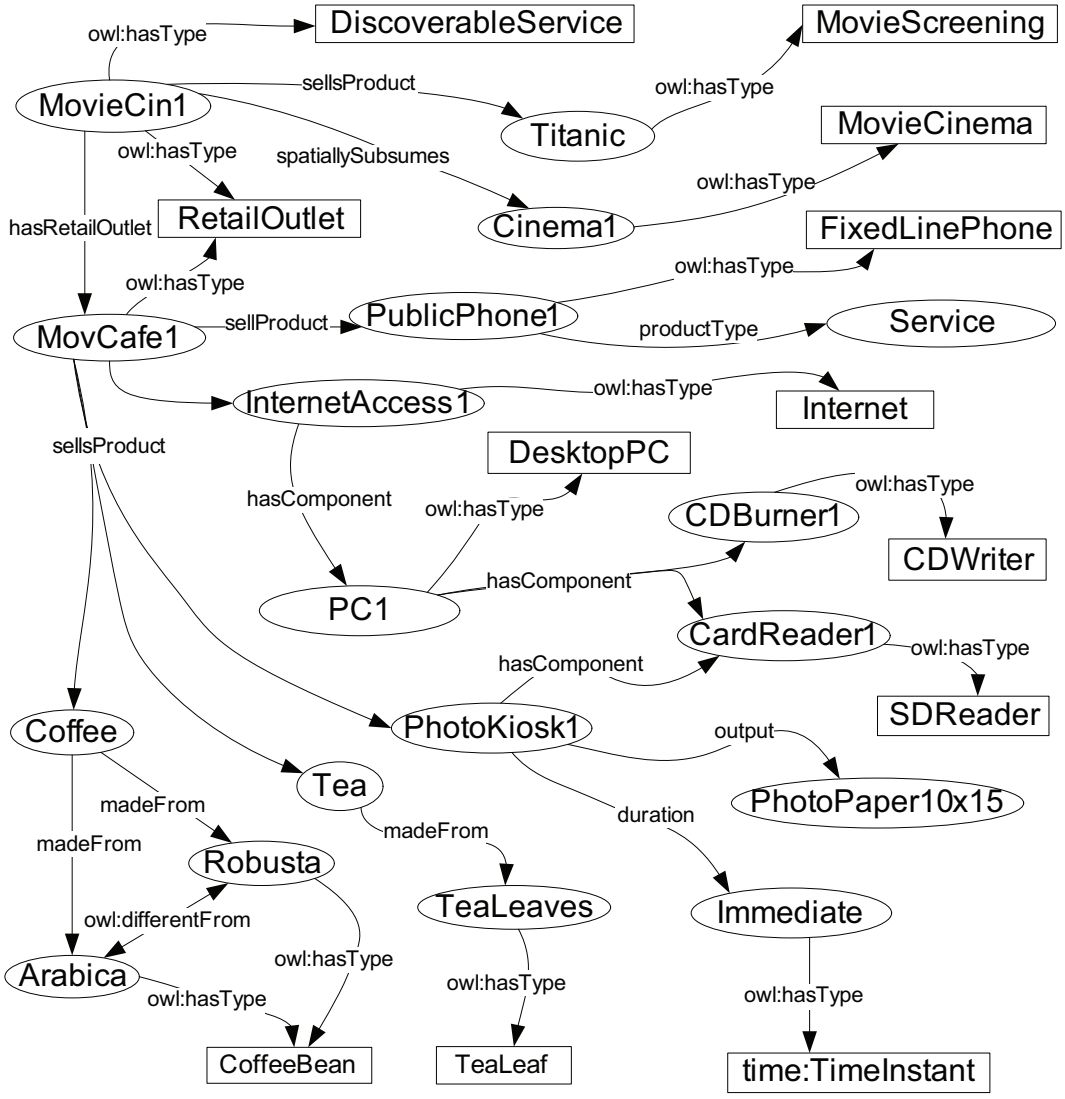


Figure 6.2: Part of the Product Case Study ABox

paper print which prints immediately. The Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) time ontology (Chen et al., 2005) is used to specify time such as TimeInstant. MovCafe1 also sells Tea and Coffee. Coffee connects to the Robusta and Arabica individuals which are both types of CoffeeBean and are specified as being different from each other (unequal). The ontologies for the product case study contain 204 classes, 241 individuals and 93 roles, which include those specified in the Device, Time, Location, Space and Geo-measurement ontologies from SOUPA (Chen et al., 2005) which the product case study makes use of.

6.3.2 Case Study 2 - Searching for a Printer

Bob walked through the main building of the university campus and his PDA connected to the university WiFi network and has downloaded the publicly available ontologies which describe the services on campus. Figure 6.3 shows one of the service descriptions which are in the ontologies that has been downloaded onto Bob's PDA. Ovals represent individuals, directed arrows represent role relations and rectangles represent class types. An individual is asserted to be a member of a class type if this individual has a relation owl:hasType to the class type.

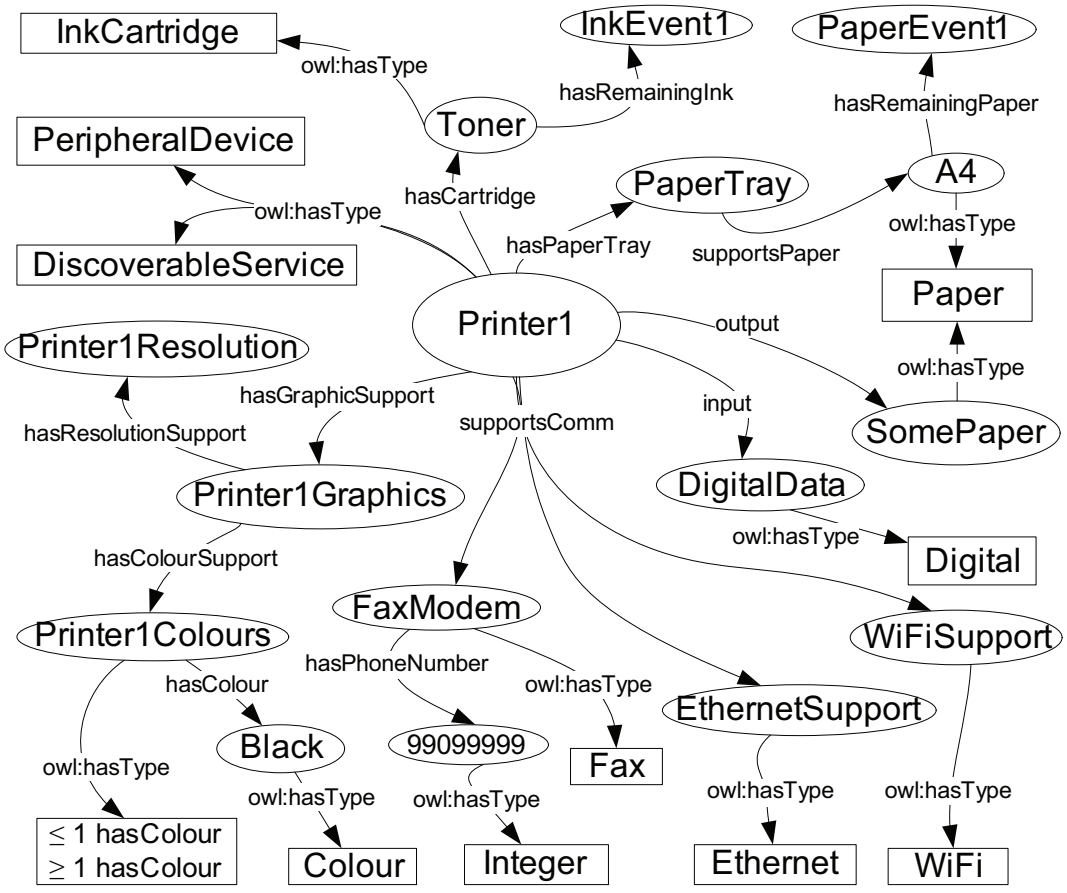


Figure 6.3: Part of the Printer Case Study ABox

In the figure, Printer1 represents a printer, which is in operation on the university campus, to which students can print. Printer1 is asserted to have the class type PeripheralDevice. The printer has various characteristics which are represented using role relations. For instance, Printer1 produces paper

based output and receives digital data as input which implies that it is a printer. Printer1 has toner in its ink cartridge implying that it is a laser printer. The toner cartridge also has notifications such as one which indicates the amount of ink remaining. The printer supports the communication protocols WiFi, Ethernet and fax. The fax protocol is supported by a fax modem, which also has a phone number implying the printer can send and receive fax documents. Printer1 has a printer tray which supports various paper sizes including A4 (the other supported types of paper are not shown). Each paper type can be associated with a paper remaining notification. Some individuals which represent notifications, such as ink and paper remaining, may also be associated with time stamps. We use the Time ontology from SOUPA (Chen et al., 2005) to specify times although this is not shown in Figure 6.3.

Printer1Graphics is used to define the graphic capabilities of Printer1. It has a role relation to Printer1Resolution which in turn has relations to various resolution metrics (not shown). Printer1Graphics also has a role relation to Printer1Colours which in turn connects to Black using the hasColour role. Printer1Colours also has the definition ≤ 1 hasColour and ≥ 1 hasColour which stipulates that Printer1Colours can only have one hasColour relation. Since Printer1Colours does have one hasColour role relation to the Black concept, it can be inferred that $\forall$ hasColour.Black is valid.

However, this could not be inferred without the cardinality restriction because OWL has an open world assumption[9]. The open world assumption is that the truth-value of a statement is independent of whether or not it is known by any single observer to be true. This means that the absence of a hasColour relation does not mean it does not exist. The ontologies used to specify the printer case study, comprise a total of 141 classes, 337 individuals and 126 roles which includes those from the Device and Time ontologies from SOUPA (Chen et al., 2005) which the printer case study makes use of.

[9]http://www.w3.org/TR/owl-guide/ (accessed May 2009)

We used the case studies outlined in this section to evaluate our light-weight mTableaux strategies. We present this evaluation in the next section. We also used these case studies to evaluate our adaptive inference strategy. We will present this evaluation in Section 6.5.

6.4 mTableaux Evaluation

In this section, we present our evaluation of the light-weight mTableaux optimisation and caching strategies. In this evaluation we aim to establish / address the following main questions:

1. How does mTableaux compare with other widely available reasoners, in terms of processing time efficiency and accuracy? We evaluate this in Section 6.4.2, using our case study ontologies as well as other publicly available ontologies;

2. How does mTableaux perform on a resource constrained mobile device, in terms of performance efficiency? We evaluate this in Section 6.4.3 using our case study ontologies.

In order to answer these questions, in the next section we will establish user requests which are matched against the service descriptions in the ontology which was presented in the previous section.

6.4.1 User Request Definitions for the Case Studies

In this section we will provide the user requests for each of the case studies from Section 6.3. These were used in our mTableaux performance evaluations which are presented later in this section.

Case Study 1: ProductRequest

Bob sits down in a park which is out of network range from the download points which he has passed. He decides that he feels like watching a movie.

However, while he is waiting for the movie he wants to use the Internet in order to lookup some accommodation, use a public phone to make bookings and he wants to download photos from the SD card in his camera and burn them to CD. As such, Bob defines a request for a movie cinema which has an Internet cafe and has a public phone. Bob wants to access the Internet using a desktop PC which supports SD card reading and CD burning. Bob defines the request definition ProductRequest1. This request was matched against service descriptions contained in the ontologies from the product case study in Section 6.3.1 in order to evaluate our light-weight inference strategy. The request ProductRequest1 and the definitions which it uses are shown in Table 6.1. We also define ProductRequest2 which contains a subset of the conditions in ProductRequest1 and was used to evaluate our mTableaux caching strategy.

ProductRequest1 is a conjunction of several class concept definitions. Each definition thus represents a specific condition of the request. Specifically, the ProductRequest1 contains several conditions / requirements. The first requirement is that the service must have RetailOutlet added as a type. The second requirement is defined in CinemaRequest which specifies that the service must sell a product which is a MovieScreening and that the service has a relation to MovieCinema using the role spatiallySubsumes which means it contains a cinema. The third condition is that Internet is available at the movie cinema or at another retail outlet contained within the movie cinema centre. Internet availability is specified as selling a product which has Internet as a class type. The Internet access should also be provided by a DesktopPC because Bob does not have his own laptop. As such, the Internet must have a component which matches the PCRequest definition. This definition is met if the product component is a DesktopPC and the desktop PC must also have the components SDReader and CDWriter so that Bob can burn his photos to CD. The fourth condition specifies that the movie cinema or other retail outlet within the movie cinema must be a cafe. A cafe is defined as anything which sells

Class Concept		Class Definition
ProductRequest1	≡	RetailOutlet ⊓ CinemaRequest ⊓ (InternetRequest ⊔ ∃ hasRetailOutlet.InternetRequest) ⊓ (CafeRequest ⊔ ∃ hasRetailOutlet.CafeRequest) ⊓ (PhoneRequest ⊔ ∃ hasRetailOutlet.PhoneRequest)
ProductRequest2	≡	(InternetRequest ⊔ ∃ hasRetailOutlet.InternetRequest)
CinemaRequest	≡	∃ sellsProduct.MovieScreening ⊓ ∃ spatiallySubsumes.MovieCinema
InternetRequest	≡	∃ sellsProduct.(Internet ⊓ ∃ hasComponent. PCRequest)
PCRequest	≡	DesktopPC ⊓ ∃ hasComponent.SDReader ⊓ ∃ hasComponent.CDWriter
CafeRequest	≡	∃ sellsProduct.(∃ madeFrom.CoffeeBean ⊔ ∃ madeFrom.TeaLeaf)
PhoneRequest	≡	∃ sellsProduct.(FixedLinePhone ⊓ ∃ productType.{Service})

Table 6.1: A listing of class definitions specifying the user requests for the product case study to evaluate our light-weight mTableaux strategies

a product made from CoffeeBean or TeaLeaf. The fifth condition is that the movie cinema or other other retail outlet inside the cinema must sell a product which is a FixedLinePhone and the productType must be a Service. Moreover, Bob does not wish to purchase a phone handset he just wants to use a phone to make some calls (i.e. a service). Note, a role filler of a existential quantifier marked with curly parentheses (e.g. {Service} indicates an individual name Service, rather than a class type.

Case Study 2: PrinterRequest

Bob is at his University campus and needs to print some documents as well as send a fax from his PDA. Therefore, he submits a request for a black

and white laser printer which has a fax modem with an active phone number and the printer must be in operation. His request was matched against the service descriptions contained in the printer case study ontologies outlined in Section 6.3.2 in order to evaluate the light-weight inference strategy. The request is defined in the class concept PrinterRequest1 which makes use of other class concepts and definitions which are shown in Table 6.2. We also define PrinterRequest2 which contains a subset of the conditions in PrinterRequest1 and was used to evaluate our mTableaux caching strategy.

Class Concept		Class Definition
PrinterRequest1	$\equiv$	$\exists$ LaserPrinterRequest $\sqcap$ $\exists$ FaxRequest $\sqcap$ $\exists$ BWRequest $\sqcap$ $\exists$ WirelessRequest $\sqcap$ $\exists$ HasInkRequest
PrinterRequest2	$\equiv$	$\exists$ LaserPrinterRequest $\sqcap$ $\exists$ FaxRequest $\sqcap$ $\exists$ WirelessRequest $\sqcap$ $\exists$ HasInkRequest
LaserPrinterRequest	$\equiv$	$\exists$ input.Digital $\sqcap$ $\exists$ output.Paper $\sqcap$ $\exists$ hasCartridge.{Toner}
FaxRequest	$\equiv$	$\exists$ supportsComm.(Fax $\sqcap$ ≥ 1 hasPhoneNumber $\sqcap$ $\exists$ hasPhoneNumber.Integer)
BWRequest	$\equiv$	$\exists$ hasGraphicSupport.($\exists$ hasColourSupport.($\forall$ hasColour.{Black}))
WirelessRequest	$\equiv$	$\exists$ supportsComm.Bluetooth $\sqcup$ $\exists$ supportsComm.WiFi $\sqcup$ $\exists$ supportsComm.IrDA
HasInkRequest	$\equiv$	$\exists$ hasCartridge.(≥ 1 hasRemainingInk)

Table 6.2: A listing of class definitions specifying the user requests for the printer case study to evaluate our light-weight mTableaux strategies

PrinterRequest1 is a conjunction of several class concept definitions. Each definition represents a specific condition / requirement of the request. The first requirement is that the service must be a laser printer and is defined in

LaserPrinterRequest. The definition of a printer is an individual which connects to the class concept Digital using the role input and the concept Paper using the role output which means it receives digital input and produces paper output. A printer is deemed a laser printer if it has a toner cartridge. The second requirement is that the printer must also be a fax machine meaning that it must support the fax communication protocol and have a phone number which is an integer, so that it can send and receive faxes. The third requirement specifies that the printer should be black and white only (i.e. the user does not wish to use a colour printer as this usually requires an extra cost). The fourth requirement is that the printer must support a wireless protocol which has been specified as any of bluetooth, WiFi or IrDA. Finally, the Bob wants to discover an operational printer which he can use (i.e. he does not with to purchase a printer from a retail store). This requirement has been specified as requiring that the printer has a remaining ink event since printers which are not in operation will not have an ink cartridge in them yet. Note, a role filler of a existential quantifier marked with curly parentheses (e.g. {Black} indicates an individual name Black, rather than a class type.

We have used the requests specified in this section to evaluate mTableaux in comparison with other widely available reasoners on a desktop PC. As discussed previously, the goal of our architecture is to support fast matching and accurate of service descriptions against user requests. Therefore, we evaluated mTableaux in terms of efficiency and accuracy compared with other widely available commercial and open source reasoners. We will presented this evaluation in the next section.

6.4.2 Comparison of mTableaux with Other Reasoners

In this section we present our evaluation which has compared mTableaux to other open source and commercial reasoners in order to establish performance and accuracy of mTableaux vis-a-vis current state of the art semantic reasoners.

This evaluation is performed on a desktop environment since many reasoners are not functional / operational on a mobile device. The main questions that we investigated are:

1. How does mTableaux compare to other reasoners in terms of response time efficiency?

2. Since the mTableaux optimisation strategies do not guarantee completeness, what is the impact of mTableaux on accuracy?

As described previously in Section 4.2, under our approach to service matching, a user request is defined using a class concept definition and a service description is defined using an individual. Checking whether the user request matches a service individual thus requires an inference check between the class concept and individual. Our mTableaux optimisation and caching strategies improve the efficiency of checking whether an individual matches a class definition by reducing the completeness of the check (thereby resulting the potential loss of some accuracy). In this section, we present our evaluation of the effectiveness of our approach, in terms of improving efficiency without substantially reducing accuracy. In our evaluation mTableaux is compared again against other currently available reasoners including FaCT++ 1.1.11[10], RacerPro 1.9.2[11] and Pellet 1.5[12].

We performed a separate comparison using each of our case studies as well as several publicly available ontologies. The publicly available ontologies used in our comparison include:

1. Galen[13] ontology which defines 2751 classes and 416 roles;

2. Tambis[14] ontology which contains 188 class concept definitions and 44 role definitions;

[10]http://owl.man.ac.uk/factplusplus/ (accessed May 2009)
[11]http://www.racer-systems.com/ (accessed May 2009)
[12]http://clarkparsia.com/pellet/ (accessed May 2009)
[13]http://www.cs.man.ac.uk/ horrocks/OWL/Ontologies/galen.owl (accessed May 2008)
[14]http://www.mindswap.org/ontologies/debugging/miniTambis.owl (accessed May 2008)

3. Koala[15] ontology which contains 23 classes and 5 roles;

4. Teams[16] ontology which contains 9 classes and 3 roles.

In each comparison we matched a class concept definition which represents the user request with a certain number individuals representing service descriptions. Some of these individuals are expected to match the request and others are not. Moreover, some individuals can be inferred to have the class concept definition as a type while others cannot. Table 6.3 presents the comparisons we performed for each ontology and the expected number of matching and non-matching individuals. For instance, for the product and the printer case studies, we defined 20 service descriptions. Three of these service descriptions match the user request while 17 do not. The specific class concept definitions from the publicly available ontologies of Galen, Tambis, Koala and Teams were selected for our tests because they are defined as being equivalent to a complex definition meaning that individuals can be inferred to be members of these classes. For example BacterialGramPositiveStainResult $\equiv$ $\{\exists$ isWithReferenceTo.(effective $\sqcap$ ...) $\sqcap$ ... $\}$. In addition we selected concepts which were as complex as possible, containing several nested conjunctions, disjunctions, cardinality and quantifier definitions. The Galen, Tambis, Koala and Teams ontologies did not contain any individuals. Therefore, we created the matching and non-matching individual for each class concept which was tested in these evaluations.

It is also noteworthy that as described previously not all individuals in an ontology represent a service description. Rather some individuals may represent particular attributes of a service description. For example, assume there is a service Cafe1 which is connected to the individual Cafe1Location by the role hasLocation. Cafe1 is a discoverable service while Cafe1Location is an attribute for Cafe1. All individuals which represent discoverable service

[15]http://protege.stanford.edu/plugins/owl/owl-library/koala.owl (accessed May 2008

[16]http://www.mindswap.org/ontologies/team.owl (accessed May 2008)

Ontology	Request	Matching Service Descriptions	Non-Matching Service Descriptions	Total Service Descriptions
Product	ProductRequest1	3	17	20
Printer	PrinterRequest1	3	17	20
Galen	BacterialGram-PositiveStainResult	1	1	2
	FailureOfCell-UptakeOfBlood-GlucoseDueToCell-InsulinResistance	1	1	2
	AcutePulmonary-HeartDisease	1	1	2
	LocalAnaesthetic	1	1	2
Tambis	small-nuclear-rna	1	1	2
	peptidase	1	1	2
Koala	MaleStudent-With3Daughters	1	1	2
	KoalaWithPhD	1	1	2
Teams	MarriedPerson	1	1	2
	MixedTeam	1	1	2
		16	44	60

Table 6.3: Listing of the class definitions and the number of matching and non-matching individuals these were compared to, for each ontology

descriptions have the type DiscoverableService and all other individuals do not. In total, Galen has 55 individuals; Tambis has 11 individuals; Koala has 15 individuals and Teams has 6 individuals. The individuals listed in Table 6.3 represent the subset which are discoverable services.

As discussed previously in Section 4.2, current reasoners perform a realisation phase. This phase completes an inference check comparing every individual in the ontology against every class concept definition in the ontology to see which class types every individual has. Therefore, realisation phase results in performing $m \times n$ inference checks where m is the number of classes and n is the number of individuals in the ontology. As discussed in Section 4.2 mTableaux does not perform realisation because we only need to compare a specific user

request definition against one service description individual, rather than comparing all individuals against all class definitions. Therefore, our performance evaluation presents two separate result values for mTableaux. One result indicates the time in seconds required to perform the inference check comparing only those individuals which represent discoverable services indicated by their membership to the class concept DiscoverableService. This involves the total number of inference checks for each ontology as shown in Table 6.3. The other evaluation result for mTableaux indicates the time in seconds required to perform full realisation with mTableaux which implies comparing all individuals against the class concept. This includes the discoverable service individuals plus all other individuals in the ontology. That is, Galen: 55 individuals; Tambis: 11 individuals; Koala: 15 individuals and Teams: 6 individuals.

The reasoner comparison evaluation was completed on a Pentium Centrino 1.82GHz computer with 2GB memory with Java 1.5 (J2SE) allocated maximum of 500MB for each experiment. All times provided in this section are computed as an average of 10 independent runs.

Figure 6.4 presents the time in seconds to perform the tests for each reasoner using the Galen ontology. In this figure mTableaux significantly outperformed the other reasoners, requiring only 0.67 seconds to perform the 8 inference checks comparing discoverable services. mTableaux with full realisation almost performed as well as FaCT++ and outperformed RacerPro. Pellet required more than 40 seconds to complete. FaCT++ was outperformed by mTableaux but performed slightly faster than mTableaux with full realisation. RacerPro was slightly slower than FaCT++.

Figure 6.5 illustrates the time in seconds to perform the tests for each reasoner using the Product ontology. In this figure mTableaux outperformed the other reasons, except for FaCT++. mTableaux with full realisation outperformed Pellet and RacerPro. RacerPro produced the slowest response-time.

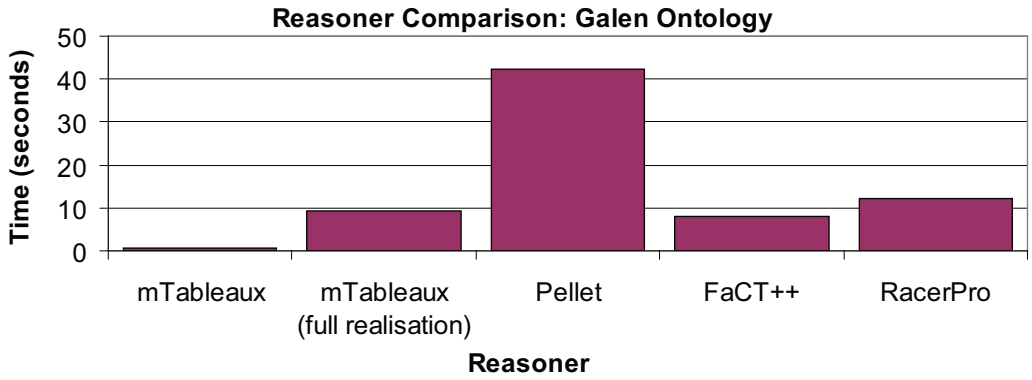


Figure 6.4: Performance comparison of mTableaux with other reasoners using the Galen ontology

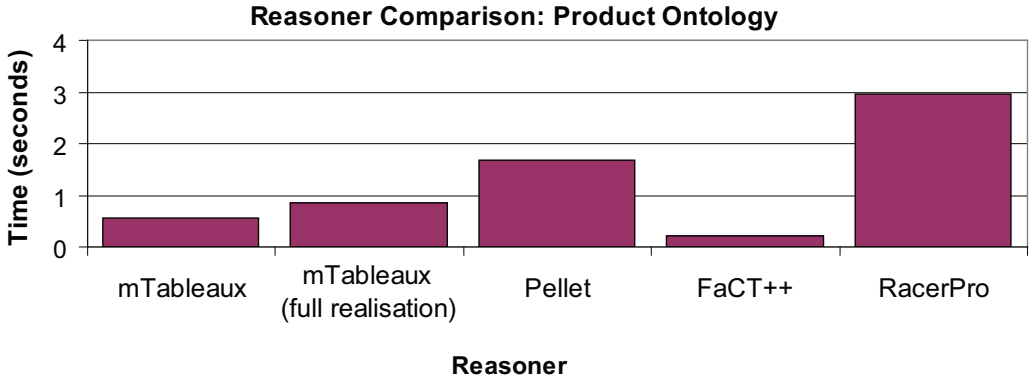


Figure 6.5: Performance comparison of mTableaux with other reasoners using the Product ontology

Figure 6.6 illustrates the time in seconds to perform the tests for each reasoner using the Product ontology. In this figure, mTableaux outperformed the other reasoners. mTableaux with full realisation out performed Pellet and RacerPro. FaCT++ could not complete the realisation task and did not provide a result. RacerPro produced the slowest response-time when compared to those reasoners which successfully produced a match result.

The inference checks for the Tambis, Koala and Teams ontologies completed in under 1 second and did not show much variation. Therefore, so we did not include these results in the figures.

As can be seen from our tests, mTableaux with no realisation performed better than mTableaux with full realisation in all cases. This is because, in mTableaux, only service description individuals are being checked against the

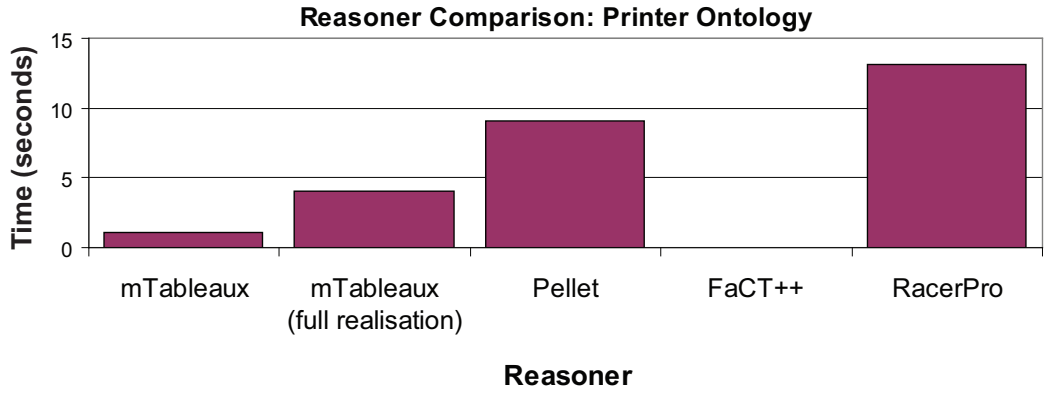


Figure 6.6: Performance comparison of mTableaux with other reasoners using the Printer ontology

user request rather than all individuals. Fewer inference checks results in improved performance. mTableaux with full realisation significantly out performed Pellet in all cases. Since mTableaux is implemented as an optimisation to the Pellet, this result shows that our optimisation strategies significantly reduce the size of the reasoning task. mTableaux with full realisation performed at least 50 percent faster than Pellet and in larger tasks such as Galen the performance improvements were significantly greater than 50 percent. We also observed that for all evaluations the number of branches applied when using mTableaux was less than half that of Pellet. We conclude that when the amount of available memory available is constrained as on a small device, the performance improvements resulting from mTableaux will be significantly enlarged. The tests showed that FaCT++ slightly out performed mTableaux with full realisation, where a result was obtained. We attribute this to the fact that FaCT++ is written in C++, which performs faster than Java which mTableaux is implemented in. However, we opted to use Java for its portability to heterogeneous small mobile devices. Furthermore, on the Printer ontology, FaCT++ did not complete.

As stated previously, the goal of our architecture is to perform accurate and fast matching of service descriptions against user requests. As shown in the tests above our optimisation strategies reduce the size of the reasoning

task and significantly improve performance. However, as stated in Section 4.2 while optimisation strategies do not guarantee completeness they aim guarantee soundness. This means that it is possible that mTableaux will product false negative matches (i.e. it may fail to successfully prove a user request matches a service description). However, mTableaux will not produce any false positives (i.e. if mTableaux proves that a request matches a service description then this result is correct). In the remainder of this subsection we will evaluate the impact which our strategies have had on accuracy. We measure accuracy in terms of precision and recall (Bernstein and Klein, 2002) of the actual service description which mTableaux found to match the user request, compared to the expected results given in Table 6.3.

Recall is a measure of whether all service descriptions which were expected to match are contained in the list of matching results. It is defined in Equation 6.1 where x denotes the number of matching service descriptions (i.e. which were expected to match) which were actually proven to match by mTableaux and n denotes the total number of matching service descriptions (i.e. which were expected to match). A recall value of 1 means that all matching service descriptions were successfully proven to match by mTableaux.

$$\text{recall} = x/n \tag{6.1}$$

Precision is a measure of whether the list of matching service descriptions returned by mTableaux contains any service descriptions which were not expected to match. Precision is defined in Equation 6.2 where x denotes the number of matching service descriptions (i.e. which were expected to match) which were actually proven to match by mTableaux and N denotes the total number of service descriptions which were actually proven to match by mTableaux (i.e. including any which were not expected to match). A precision value of 1 means that mTableaux did not incorrectly prove that any non-matching service descriptions matched the user request.

$$\text{precision} = x/N \tag{6.2}$$

In Table 6.4 we present the recall and precision results which were obtained by completing the queries related to the Product, Printer, Galen, Tambis, Koala and Teams ontologies as defined earlier in this section in Table 6.3.

Reasoner	Actual Positive	Actual Negative	Recall	Precision
mTableaux	16	44	16/16 = 1.0	16/16 = 1.0
Pellet	16	44	16/16 = 1.0	16/16 = 1.0
RacerPro	16	44	16/16 = 1.0	16/16 = 1.0
FaCT++	15	45	15/16 = 0.937	15/15 = 1.0

Table 6.4: mTableaux Recall and Precision Results

The results show that the actual results were as expected for all reasoners except that FaCT++. This means that mTableaux successfully proved a match result for all 16 of the service descriptions which were expected to match. In addition, mTableaux did not prove a match for any non-matching service descriptions. The recall and precision results were the same for Pellet and RacerPro. FaCT++ failed to successfully match one of the positive 16 service descriptions against the user request. Specifically, it could not match the class concept MaleStudentWith3Daughters in the Koala ontology, to the matching individual. This was because FaCT++ 1.1.11 does not match Boolean literal values which were present in the request class type.

These results show that although mTableaux does not theoretically guarantee completeness for its optimisation strategies, in practise it did not produce a significant reduction in result accuracy. In fact it did not produce any reduction in accuracy. mTableaux did not fail to prove any of the matching service descriptions to the user request (i.e. it did not reduce recall). In addition, mTableaux did not prove a match that any service descriptions which did not match the user request (i.e. it did not reduce precision). Therefore,

we conclude in data sets representing realistic scenarios such as our case studies presented in this section, mTableaux does not reduce result accuracy as measured by recall and precision.

In this section we have demonstrated that:

1. mTableaux is more efficient than other open source and commercial such as the RacerPro and Pellet reasoners. It performs comparatively with FaCT++ when full realisation is used and performs faster than FaCT++ when full realisation is not used;

2. In our tests accuracy as measured by recall and precision was not reduced by mTableaux. Therefore, we conclude that mTableaux does not significantly reduce accuracy in realistic data sets such as those in our evaluation;

We have established that mTableaux outperformed all reasoners except for FaCT++ in some case, while preserving completeness in our case studies. Therefore, in the next section we present a performance evaluation to show how mTableaux performs on a small resource constrained device. We also show which strategies work best together and the level of overhead incurred by using each mTableaux strategy.

6.4.3 Mobile Performance Evaluation of mTableaux

In this section, we evaluate our mTableaux optimisation and caching strategies on a small resource constrained mobile device in order to establish the following questions:

1. Does mTableaux enable successful completion of a matching task, such that a result can be obtained (i.e. was available memory was not exceeded)?

2. Does mTableaux significantly improve performance compared to standard Tableaux without our optimisation and caching strategies?

3. Do our optimisation and caching strategies result in significant overhead in terms of processing time?

4. When a request is matched against a matching and a non-matching service description do the optimisation strategies improve processing time efficiency in both cases?

5. Do different strategies work better for different case studies / inference tasks? Which mTableaux strategies or combination of strategies work best?

We used the Product and Printer ontologies from Section 6.3 to evaluate our mTableaux optimisation and caching strategies in order to investigate the above questions.

Table 6.5 lists the user request to service description comparisons used in this section as well as the expected result for each comparison. The comparisons involving ProductRequest1 and PrinterRequest1 were performed on a mobile device and the response time was evaluated as presented later in this section. The comparisons involving ProductRequest2 and PrinterRequest2 were stored in the cache which was used in some tests to evaluate the caching strategy (i.e. in the tests where the caching strategy was enabled).

The requests ProductRequest1 and ProductRequest2 from the product case study were defined in Section 6.4.1. ProductRequest1 is a request for a movie cinema with Internet cafe. ProductRequest2 is a subset of ProductRequest1 containing only the requirement for Internet access. An excerpt of the product case study ontologies was presented in Section 6.3.1, which contained a service description MovieCin1 that completely matches ProductRequest1 and ProductRequest1. In addition, the product ontologies also contain a service description MovieCin2 which does not match the requests ProductRequest1 or ProductRequest2 because the desktop PC in the Internet cafe does not have an

Case Study	Request	Service Description	Match / No Match
1	ProductRequest1	MovieCin1	Match
		MovieCin2	No Match (no SD card reader)
	ProductRequest2	MovieCin1	Match
		MovieCin2	No Match (no SD card reader)
2	PrinterRequest1	Printer1	Match
		Printer2	No Match (no ph. number)
	PrinterRequest2	Printer1	Match
		Printer2	No Match (no ph. number)

Table 6.5: A listing of four user request to service description comparisons and whether or not they match

SD card reader, which is a requirement in the requests. Note, the individual MovieCin2 was not shown in the excerpt presented in Section 6.3.1.

The requests PrinterRequest1 and PrinterRequest2 from the printer case study were defined in Section 6.4.1. PrinterRequest1 is a request for a laser printer fax machine. PrinterRequest2 is a subset of PrinterRequest1 since it leaves out the requirement for black and white printing. An excerpt of the printer case study ontologies was presented in Section 6.3.2, which contained a service description Printer1 that completely matches PrinterRequest1 and PrinterRequest2. In addition, the printer ontologies contain a service description Printer2 which does not match the requests PrinterRequest1 or PrinterRequest2 because the fax machine modem does not have a phone number, which is a requirement in the the requests. Note, the individual Printer2 was not shown in the excerpt presented in Section 6.3.2.

In order to evaluate which optimisation strategies work best together we repeated each of the four match checks from Table 6.5 with different combinations of the optimisation and caching strategies enabled. Table 3.5.2 presents 12 different tests which each have a different combination of strategies enabled.

As described previously in Section 4.2, our light-weight matching approach incorporates two optimisation strategies. These include the selective application of transformation rule (ST) and selective application of the disjunction transformation rule (SD). Our approach also includes a caching strategy (CS).

The caching strategy works in one of two ways which we denote CSa and CSb. These modes reflect whether or not particular entries in the cache need to be re-evaluated or not, as was detailed in Section 4.5. However, we provide a re-cap as follows. A request is made up of many conditions and subconditions. Where a request is being matched against a service description and an entry is found in the cache for a condition or sub-condition of the request, this cache entry will represent a previous complete or partial match result for this condition. The entry will also have a time stamp associated with it. mTableaux will utilise the cache entry for the request condition in one of two ways:

a. if the entry represents a previous complete match for the condition and has a time stamp which does not exceed a user specified expiry time associated with the request condition, then it can be used to immediately generate a match for the condition, without requiring a re-evaluation of this condition;

b. if the entry represents a previous partial match, or the time stamp associated with the cache entry exceeds the user specified expiry time associated with the request condition, then this condition must be re-evaluated. However, the evaluations which proved the match previously, will be prioritised so that they are evaluated first.

Since the caching strategy works in one of these two modes, we will evaluate each of these separately. CSa implies that an immediate match will be generated for entries which represent a complete match. CSb implies that the request condition will need to be re-evaluated but the evaluations which proved the match previously will be applied first.

The CS strategy can function in CSa or CSb mode but not both (if an entry generates an immediate clash for a request condition then this implies

it is not re-evaluated). Therefore, the 12 tests in Table 3.5.2 represent all possible combinations of mTableaux optimisation and caching strategies. Test 12 represents normal execution of the standard Tableaux algorithm with none of our optimisation or caching strategies enabled.

Test Number			1	2	3	4	5	6	7	8	9	10	11	12
Selective Transformations		(ST)	✓		✓		✓		✓		✓		✓	
Selective Disjunctions		(SD)	✓	✓			✓	✓			✓	✓		
Cache	Immediate Match	(CSa)	✓	✓	✓	✓								
	Re-Order Evaluations	(CSb)					✓	✓	✓	✓				

Table 6.6: A listing of twelve tests, each with a different combination of our mTableaux optimisation and caching strategies enabled

Finally, in addition to evaluating the two caching strategy modes we also evaluate each of these modes when either the entire request is cached or only part of the request is in the cache. Therefore, whenever the caching strategy is enabled two separate evaluations were conducted where:

1. the match results for the entire request and service description are stored in the cache;

2. the match results for a *subset of the* request and the service description are stored in the cache.

In the remainder of this section, we use mTableaux to compare a request against a matching and a non-matching service description, for each case study, as shown in Table 6.5.

Our evaluations were performed on a HP iPAQ hx2700 PDA, with Marvell PXA270 624MHz processor, 256MB memory (64MB main memory, 192MB flash ROM) with the Windows Mobile 5.0 operating system. We consider evaluations in this device to be compariable with currently available small mobile devices such as smart phones and PDAs. For instance, currently available

HP smart phones include the HP iPAQ 612c[17] and HP iPAQ 912c[18] which use the same processor (at slower speeds of 520MHz and 416MHz, respectively), as the device we performed our evaluations on. Currently available HP PDAs include the HP iPAQ 112[19] and HP iPAQ 212[20], which have 624MHz processors and 64MB and 128MB main memory, respectively.

Our evaluations are performed using the Mysaifu[21] Java J2SE Virtual Machine (JVM). Since memory on the device is used for both the operating system and running programs, the Java virtual machine was allocated 15MB of memory for our tests. As discussed in Section 6.2, mTableaux is implemented in the Pellet reasoner. In all of our tests, Pellet was running with $\mathcal{SHOIN}$ expressiveness.

We attempted some of these inference checks on a mobile device using the standard Pellet reasoner. However, as was shown in Figure 1.2 in Section 1.2 the device ran out of memory during the initial consistency check which is performed by current standard reasoners.

We performed the evaluations of mTableaux, corresponding to the tests from Table 3.5.2 with different combinations of the optimisation and caching strategies enabled. The strategies which were enabled for each test are listed above each test on the graph. Some tests did not complete because there was insufficient memory to complete them. This is also indicated in the bars on the graphs. It is noteworthy to mention that in all tests the Java virtual machine (JVM) always used all of the memory allocated.

[17] http://h10010.www1.hp.com/wwpc/au/en/sm/WF05a/215348-215348-64929-3352590-3352590-3544356.html (accessed Dec 2009)

[18] http://h10010.www1.hp.com/wwpc/au/en/sm/WF05a/215348-215348-64929-3352590-3352590-3551668.html (accessed Dec 2009)

[19] http://h10010.www1.hp.com/wwpc/au/en/sm/WF05a/215348-215348-64929-215384-215384-3544250.html (accessed Dec 2009)

[20] http://h10010.www1.hp.com/wwpc/au/en/sm/WF05a/215348-215348-64929-215384-215384-3544059.html (accessed Dec 2009)

[21] http://www2s.biglobe.ne.jp/ dat/java/project/jvm/index_en.html (accessed May 2009)

Figure 6.7 presents the response time for matching the request ProductRequest1 against the service description Product1 which was found to successfully match the request in all tests which completed.

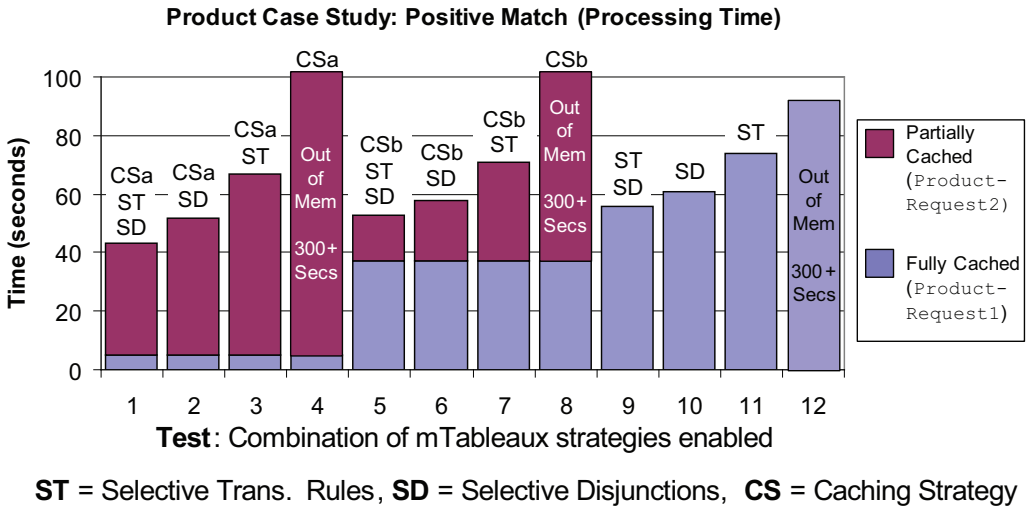


Figure 6.7: Time in seconds required to perform matching of request ProductRequest1 with service description MovieCin1, for each test

The performance of the optimisation strategies can be seen by looking at Tests 9-12 where the caching strategy was not enabled. Note for Tests 9-12 the response-time was the same when the request was both partially cached (i.e. when ProductRequest2 was cached) and fully cached (i.e. when ProductRequest1 was cached) because the cache is not used in these tests. Test 12, which had no optimisation strategies enabled ran out of memory and failed to complete after executing for over 300 seconds. Conversely, in Test 9, which had both the SD and ST optimisation strategies enabled, with no cache, the matching process required 56 seconds to complete. The SD strategy proved to be more effective than the ST strategy by comparison in Tests 10 and 11, but using both provided the best result as shown in Test 9.

In terms of the caching strategy, when the comparison of ProductRequest1 against Printer1 was previously performed (i.e. the results were stored in the cache), performing the match check for the same request and service description with CSa enabled, produced an immediate clash for all request conditions,

thus, producing a result in less than 5 seconds as shown in Tests 1-4. Alternatively, Tests 5-8 show performance results when CSb was enabled (and results from Printer1 cached) which requires a re-evaluation of the cached request requirements but prioritises those which generated a match previously. Tests 5-8 showed significant performance gains by using this prioritisation (compared to tests 9-12 where no caching was enabled). In addition, Tests 5-8 all completed with the same response time, which implies that it did not make any difference which other optimisation strategies were enabled, if the request is previously cached and CSa or CSb is enabled. Comparing Tests 5-8 to Test 9 shows that applying only those evaluations in the reasoner which generate a match (derived from the cache) is more efficient than our optimisation strategies. The extra evaluations which led to a slower response time when comparing Test 5 (fully cached) against Test 9 was due to the fact that MovCin1 had an Internet cafe which sold 30 products. This meant that each of the 30 products would need to be searched when matching particular request requirements such as the requirement for selling Tea or Coffee, which could not be avoided even when using our ST or SD strategies. Using the cache avoided this additional search.

Alternatively, when ProductRequest2 was cached, this still improved response time when CSa was enabled, compared to without CSa. This can be seen by comparing Tests 1-4 (with patial cache) against Tests 9-12. Although the CSa strategy improved response time, the improvement was not significant as when ProductRequest1 was fully cached since ProductRequest2 only contains one of the 5 requirements which ProductRequest1 contains. When using the CSb strategy (with partial cache), there was a slight performance improvement compared to without CSb enabled which can be seen when comparing Test 5-8 with Tests 9-12. Tests 4 and 8 failed to complete (with partial cache) because checking the request requirements which were not contained in the cache was

still too large to complete without the optimisation strategies enabled (i.e. with ST and SD disabled).

Figure 6.8 presents the performance results when using mTableaux to match the request ProductRequest1 against the service description MovieCin2 which does not match this request.

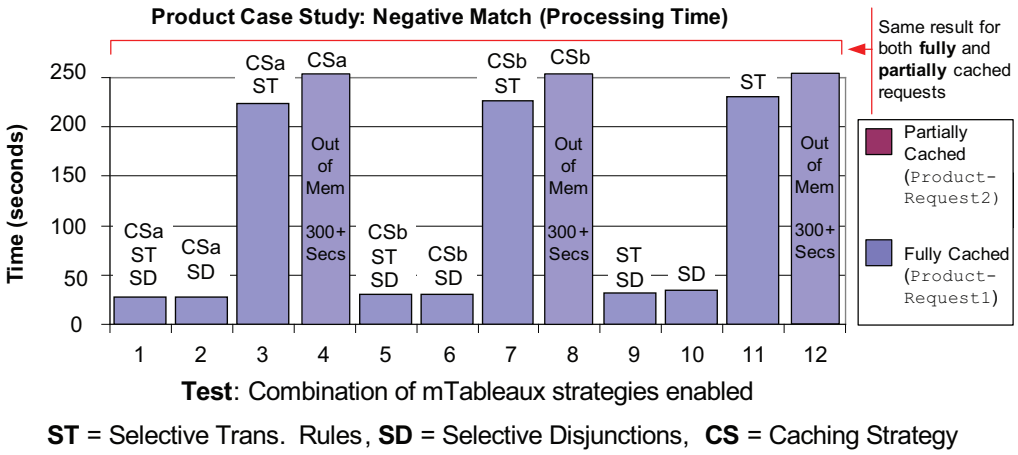


Figure 6.8: Time in seconds required to perform matching of request ProductRequest1 with service description MovieCin2, for each test

The performance of the optimisation strategies can be seen by looking at Tests 9-12 where the caching strategy was not enabled. Note for Tests 9-12 the response time was the same when the request was both partially cached (i.e. when ProductRequest2 was cached) and fully cached (i.e. when ProductRequest1 was cached) because the cache is not used in these tests. In Test 12, when no optimisation strategies were enabled, a result could not be obtained after 300 seconds. Alternatively, in Test 9, which had both the ST and SD strategies enabled, the matching process was completed in 32 seconds. The SD strategy was more effective than ST, providing a very similar result to when both SD and ST were enabled (see Tests 9 and 10). While the ST strategy improved the response time in Test 11 compared to when no strategy was enabled in Test 12, ST did not perform anywhere near as well as the SD strategy for this scenario.

In Tests 1-8 the response time was the same when the request was both partially cached (i.e. when ProductRequest2 was cached) and fully cached (i.e.

when ProductRequest1 was cached) because although the cache was enabled, no cache entries were found to improve response time. The reason for this was that the requirement for a desktop PC with secure digital card reader (which did not match) was checked very early in the matching process and the cache only stores positive matches (not negative ones). Since the service description MovCin1 does not have a secure digital card reader the subsequent request requirements, where the cache may have helped, were not checked because the match had already failed. The fact that the check failed early in the matching process also resulted in the caching strategy providing no significant improvement in response time when it was operating in with CSa (see Tests 1-4) or CSb (see Tests 5-8) modes, compared to not having the caching strategy enabled (see tests 9-12). Most of the processing time was used to prove that the service description MovCin1 failed to match the requirement for a card reader. Furthermore, Tests 4 and 8 failed to complete (for both the partially and fully cached request) because checking the request requirements which were not stored in the cache was too large to complete without the optimisation strategies enabled (i.e. with ST and SD disabled).

Figure 6.9 presents the performance results when using mTableaux to match the request PrinterRequest1 against the service description Printer1, matches the request.

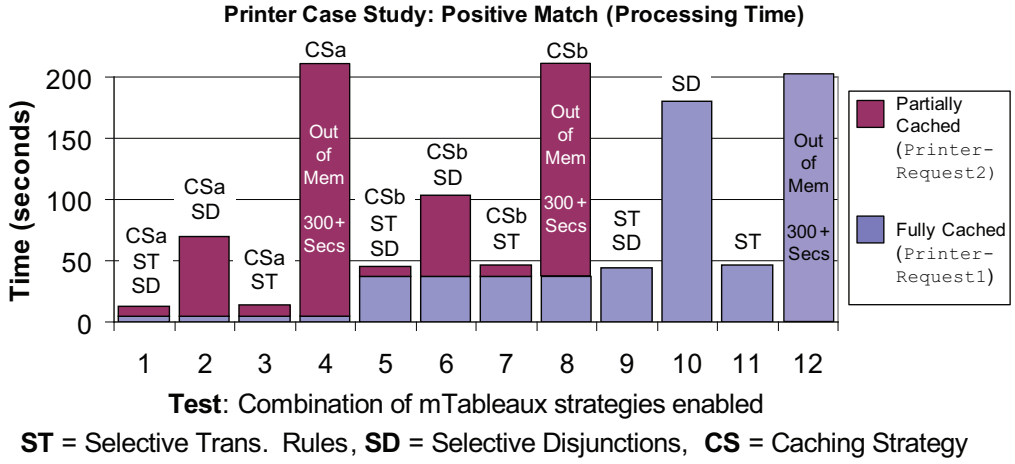


Figure 6.9: Time in seconds required to perform matching of request Printer-Request1 with service description Printer1, for each test

The performance of the ST and SD optimisation strategies can be seen by looking at Tests 9-12 where the caching strategy was not enabled. Note for Tests 9-12 the response-time was the same when the request was both partially cached (i.e. when PrinterRequest2 was cached) and fully cached (i.e. when PrinterRequest1 was cached) because the cache is not used in these tests. In Test 12, when no strategies were enabled, no result was obtained after 300 seconds. Alternatively, when both ST and SD strategies were enabled in Test 9, the response time was 44 seconds. The ST strategy was more effective than the SD strategy for this scenario which is shown by comparing Tests 10 and 11. Enabling both SD and ST (see Test 9) did not provide a major performance improvement compared to using only ST (see Test 11).

When the CSa caching strategy was used, which implies an immediate clash can be generated for cached request requirements, there was a significant improvement in response time when the request was fully cached (compared to Test 9-12 where no cache was enabled). Tests 1-4 (with full cache) had a response time of under 5 seconds. When CSb was used in Tests 5-8 (with full cache), the response time was identical for all of these tests because the ST and SD strategies could not further reduce the number of evaluations. Tests 5-8 (with the full cache) had a response time of 36 seconds compared with Test 9 (with no cache) which had a response time of 44 seconds. Thus, CSb resulted in an improvement (with full cache) but this was not as significant as the improvement for the product case study, positive test. Thus, the ST and SD strategies were more effective in reducing the size of the matching problem with the product case study. This was primarily because the Printer1 service description had fewer search options for universally / existentially quantified role relations (arising from the request PrinterRequest1) compared to MovCin1 in the product case study which was selling 30 products to search through.

When the CSa caching strategy was used with PrinterRequest2 in the cache (meaning PrinterRequest1 was partially cached), the CSa strategy proved very

effective. This was because only the black and white request requirement had to be evaluated and all others contained in the cache, thereby generating an immediate clash. When using the CSb caching strategy (with partial cache) as shown in Tests 5-8, the response time was about the same when ST was also enabled (compare Test 5 with 9 and Test 7 with 11). Comparing Test 6 (with partial cache) to 10 shows that the SD strategy was more effective for reducing the size of the matching problem when checking the black and white requirement which was not cached, than it was for reducing the size of the problem for matching all requirements. Test 4 and 8 (with partial cache) failed to complete because the size of the task required to check the black and white requirement (which was not cached), was too large to complete without the optimisation strategies enabled (i.e. with ST and SD disabled).

Figure 6.10 presents the performance results when using mTableaux to match the request PrinterRequest1 against the service description Printer1 which does not match the request.

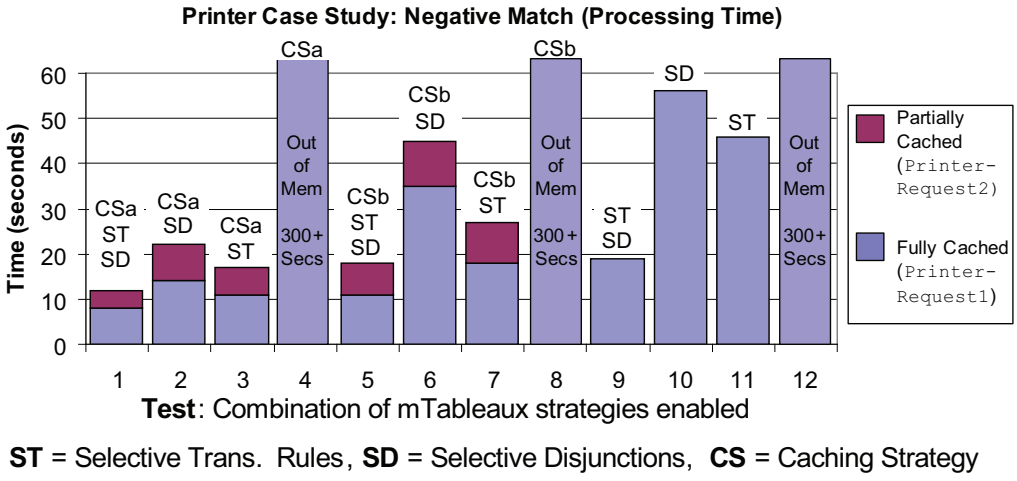


Figure 6.10: Time in seconds required to perform matching of request Printer-Request1 with service description Printer2, for each test

The performance of the optimisation strategies can be seen by looking at Tests 9-12 where the caching strategy was not enabled. Note for Tests 9-12 the response-time was the same when the request was both partially cached (i.e. when PrinterRequest2 was cached) and fully cached (i.e. when PrinterRequest1

was cached) because the cache is not used in these tests. Test 12, which had no optimisation strategies enabled, ran out of memory and failed to complete after executing for over 300 seconds. Conversely, in Test 9, which had both optimisation strategies ST and SD enabled (but not caching strategy CS), the response time was 19 seconds. The ST strategy proved to be slightly more effective than the SD strategy by comparison of Tests 10 and 11. Using ST as well as SD (see Test 9) provided a performance improvement when compared to using ST alone (see Test 11).

When the caching strategy was used in CSa mode in Tests 1-4, where the request PrinterRequest1 was fully cached, performance improvements were observed even though the request PrinterRequest1 does not match the service Printer2. The reason was that Printer2 did match all requirements in PrinterRequest1 except for the requirement for a phone number and the phone number was not checked until late in the matching process. Therefore, in Test 1 when (with full cache), all conditions except for the phone number requirement, generated an immediate clash without needing to be re-evaluated. Thus, the reasoner only needed to check the phone number requirement which then failed to match. The phone number requirement in PrinterRequest1 was not stored in the cache because the cache only stores positive match results. The variance in response time for Tests 1-4 (with full cache) was a reflection of how effectively the optimisation strategies reduced the size of the problem to check the phone number requirement (which was not in the cache). When the caching strategy was in CSb mode (with full cache), it required re-evaluation of all cached requirements. Therefore, the performance of CSb (see Tests 5-8 with full cache) was slower than CSa (see Tests 1-4 with full cache). Like for Tests 1-4 (with full cache), the variance in response time for Tests 5-8 (with full cache) was caused by how effective the optimisation strategies were in reducing the size of the problem to check the phone number requirement (which was not in the cache). Test 4 and 8 (with full cache) failed to complete because the size of

the task required to check the black and white requirement, was too large to complete without the optimisation strategies enabled (i.e. with ST and SD disabled).

In Tests 1-4, where the PrinterRequest2 request was cached (meaning PrinterRequest1 was partially cached), the black and white requirement also had to be checked. The variance between Tests 1-4 (with partial cache) was a reflection of the effectiveness of the optimisation strategies which were enabled in each test. This is also the case for Tests 5-8 (with partial cache). The CSa strategy improved response time which can be seen by comparing Tests 1 (with partial cache) with 9. The CSb strategy slightly improved response time, when comparing Test 5 (with partial cache) and 9. However, the improvement was marginal because of the required re-evaluation of the requirements in the request.

The above positive and negative tests using the product and printer case studies have shown that our optimisation strategies significantly reduce the processing time required to perform a matching check using a Tableaux reasoner. In fact, without our strategies enabled, a result could not be obtained. In addition, when the SD strategy was used in isolation, it performed better than ST for the product case study 1. Alternatively, the ST strategy performed better than SD for the printer case study 2. However, the best response time was achieved in all cases when both the ST and SD strategies were used together. The caching strategy also improved response time when used in either CSa or CSb mode. However, the CSa produces better response times than CSb because CSa does not require re-evaluation of cached entries. The caching strategies were effective even when only part of the request was cached but when more request requirements were in the cache this led to more substantial improvements in response time. The optimisation and cache strategies were effective in improving response times for negative as well as positive

match checks. In the remainder of this section, we will evaluate the response time overhead which was created by these optimisation and caching strategies.

Figures 6.11, 6.12, 6.13 and 6.14 present the overhead in processing time for each of the optimisation and caching strategies which were enabled in each of the tests presented earlier in this section. Where the caching strategy was enabled the overhead results represent those which were incurred when the request was fully cached.

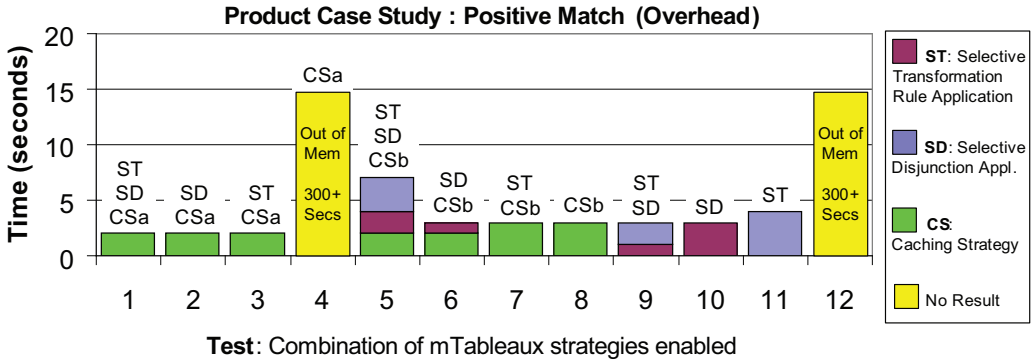


Figure 6.11: Overhead processing time in seconds incurred by using the optimisation and caching strategies enabled for each test when matching ProductRequest1 against MovieCin1

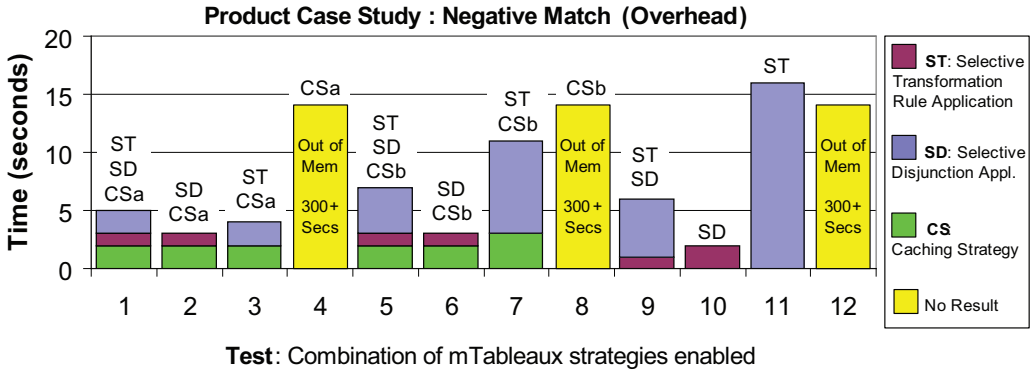


Figure 6.12: Overhead processing time in seconds incurred by using the optimisation and caching strategies enabled for each test when matching ProductRequest1 against MovieCin2

The figures show the level to which each strategy contributes to the total overhead for the test. In some tests, some strategies did not contribute any overhead. For instance, when a match check was performed on the service

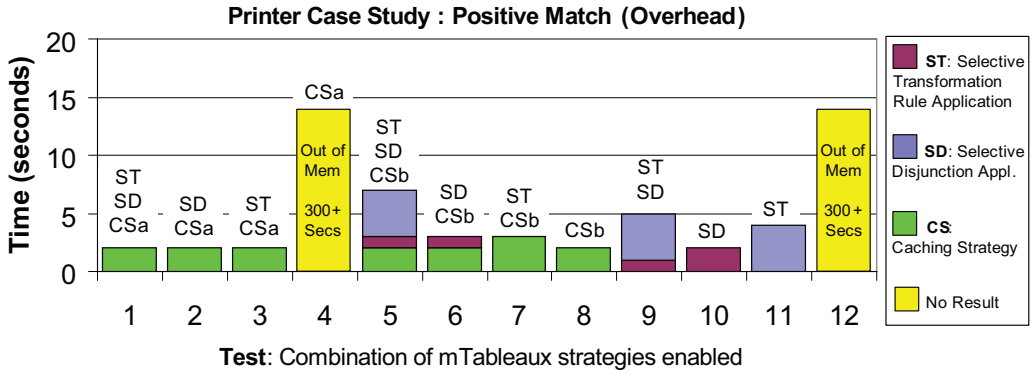


Figure 6.13: Overhead processing time in seconds incurred by using the optimisation and caching strategies enabled for each test when matching Printer-Request1 against Printer1

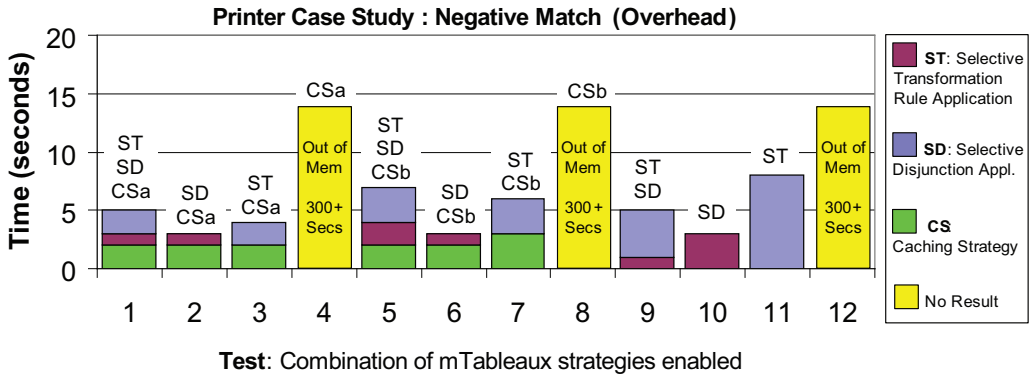


Figure 6.14: Overhead processing time in seconds incurred by using the optimisation and caching strategies enabled for each test when matching Printer-Request1 against Printer2

descriptions which did match the user request (see Figures 6.11 and 6.13), only the caching strategy contributed to the overhead for Tests 1-3. The reason for this was that an immediate match was generated by the caching strategy operating in CSa mode, for these service descriptions. Therefore, no Tableaux rules were evaluated / applied meaning that the ST and SD strategies were not used. Alternatively, for the match checks comparing service descriptions which did not match the request (see Figures 6.12 and 6.14), Tableaux rules were applied to check the request condition which did not match the service descriptions, which was not contained in the cache. Therefore, the SD and ST strategies were used when checking that request condition.

We observed that our optimisation and caching strategies use very little overhead. The selective disjunction transformation (SD) optimisation incurred the lowest overhead of around 1-2 seconds for all tests where it was enabled. This represented approximately 0.07% of the response time for the fastest response time and approximately 0.02% for the slowest response time, where ST and SD were enabled. The selective transformation application (ST) optimisation generally required 2-3 seconds in all optimisation strategies with the exception of Test 11 when a negative match was performed. This represented approximately 0.13% of the total time for the fastest response time and approximately 0.04% for the slowest response time, where ST and SD were enabled. ST had an overhead of 16 seconds for the negative match check in the product case study (see Figure 6.12) and 9 seconds for the negative match check in the printer case study (see Figure 6.14). We attribute this to the fact that in Test 11 the ST optimisation is being used without any other optimisation or caching strategy enabled. Also, when a negative match is performed the ST must exhaustively search for and add all individuals to the set which can be evaluated before it can stop matching and declare that the service description did not meet the user request. This overhead was reduced when the SD optimisation was also enabled motivating the need to enable both the ST and SD optimisation strategies.

In this section we have demonstrated through our extensive evaluation and analysis that:

1. mTableaux reduces the size of the inference problem by reducing the number of expansions performed by the reasoner. Our evaluations show that mTableaux successfully enables the completion of a matching task on a small, resource constrained device without exceeding available memory. Conversely, without the strategies enabled the task could not be completed;

2. mTableaux significantly improves the response time to perform matching when the optimisation and caching strategies were enabled compared to when these were not enabled. In fact without the strategies enabled the task could not be completed;

3. mTableaux optimisation and caching strategies used together has minimal extra processing overhead;

4. mTableaux strategies improved performance for inference checks which both provided a positive and a negative match result;

5. the effectiveness of each optimisation strategy depends on the ontology which matching is being performed on. Our evaluations showed that the ST optimisation strategy was more effective in improving efficiency in the first case study while SD was more effective in the second case study. Utilising all strategies together provided the fastest / most efficient response-time.

In this section we have shown that our mTableaux strategies, which were proposed in Chapter 4, successfully optimise semantic reasoning to enable efficient light-weight matching on a mobile device. In the next section we present our evaluation of the adaptive inference strategy, which was proposed in Chapter 5.

6.5 Adaptive Inference Strategy Evaluation

In this section we present our evaluation of our adaptive inference strategy. This strategy is designed to enable interruption of the matching process in the case that user constraints such as available time or resources are exceeded. The match result then takes into consideration the intermediate processing which has been completed. In this section we show that our strategy effectively meets these aims using time as the constraint to interrupt the matching

process. However any metric can be used by our proposed adaptive strategy. Specifically, in our evaluation we show that our strategy addresses the following main questions:

1. Current reasoners require that the entire matching process be completed under an "all or nothing" principle in which all requirements in the user request must be checked otherwise no match result is provided to the user. Does our adaptive inference strategy support anytime matching which means that the matching process can be interrupted anytime, depending on resource or time constraints?

2. Current reasoners match conditions in the user request in an arbitrary / depth-first order. Does our adaptive inference strategy support priority based matching, meaning that the most important request conditions, as deemed by the user, are matched first?

3. Current reasoners provide a failed match result if any requirement in the user request fails to match, even if this requirement is unimportant to the user. Does our adaptive inference strategy support partial matching, where service descriptions do not completely match a user request?

In order to answer these questions, in the next section we first establish weighted requests which take into account the importance of each condition in the user request and establish several different service descriptions to match the requests against, for each of the two case studies described in Section 6.3.

6.5.1 Weighted User Request Definitions and Service Descriptions for the Case Studies

In this section we will provide several user requests with weighted conditions as well as several service descriptions which either completely match a request, partially match a request or fail to match a request, for the product and printer case studies described in Section 6.3.

Product Case Study

Bob is in a foreign city and his mobile device has downloaded various ontologies as he has passed by download points and shop fronts. Bob wants to watch a movie, but while he is waiting for the movie to start he wants to use the Internet. As such, Bob defines a request for a retail outlet, which is a movie cinema, has an Internet cafe. However, different features / requirements have a different level of importance to Bob. For instance, the most important feature to Bob is that the service description be about a retail outlet, the next important requirement is that the service be a Movie cinema. Internet access is less important, but still more important than being able to buy some coffee. This gives rise to ProductRequest1, in Table 6.7 (which contains similar definitions to the non-weighted ProductRequest1 from Section 6.4.1). As shown in the table, each request condition which is defined as conjunctive concept in the definition, is associated with a normalised weight value, between 0 and 1. We have also defined three different requests listed in Table 6.7, which leave out some requirements but incorporate the additional requirement for photo printing which is defined as a service which outputs photo paper, has an SD card reader and produces the photos immediately. Note there are two separate photo printing definitions used in different requests. The only difference between these is that they have different weight values of user importance. Note also, a role filler of a existential quantifier marked with curly parentheses (e.g. {PhotoPaper10x15} indicates an individual name PhotoPaper10x15, rather than a class type. In total we have four requests and these make use of additional weighted class definitions which are defined in Table 6.8.

In our evaluations we will compare the request ProductRequest1 against eight service descriptions which are given in Table 6.9. One service completely matches ProductRequest1, six services partially match ProductRequest1 and one service does not match any condition in ProductRequest1. The degree of match for each service to the request ProductRequest1 is also given in the

Class Concept		Weight	Class Definition
ProductRequest1	≡	(1.0)	RetailOutlet ⊓
		(0.9)	CinemaRequest ⊓
		(0.7)	(InternetRequest ⊔
			∃ hasRetailOutlet.InternetRequest) ⊓
		(0.4)	(CafeRequest ⊔
			∃ hasRetailOutlet.CafeRequest) ⊓
ProductRequest2	≡	(0.9)	RetailOutlet ⊓
		(0.7)	CinemaRequest ⊓
		(0.4)	(InternetRequest ⊔
			∃ hasRetailOutlet.InternetRequest) ⊓
		(0.1)	(CafeRequest ⊔
			∃ hasRetailOutlet.CafeRequest) ⊓
		(0.1)	(PhotoRequest ⊔
			∃ hasRetailOutlet.PhotoRequest1)
ProductRequest3	≡	(1.0)	(InternetRequest ⊔
			∃ hasRetailOutlet.InternetRequest) ⊓
		(0.6)	(PhotoRequest ⊔
			∃ hasRetailOutlet.PhotoRequest1)
		(0.4)	(CafeRequest ⊔
			∃ hasRetailOutlet.CafeRequest)
ProductRequest4	≡	(1.0)	(InternetRequest ⊔
			∃ hasRetailOutlet.InternetRequest) ⊓
		(1.0)	(CafeRequest ⊔
			∃ hasRetailOutlet.CafeRequest)
		(0.6)	(PhotoRequest ⊔
			∃ hasRetailOutlet.PhotoRequest2)

Table 6.7: A listing of class definitions representing four user requests and explicit weights which express the importance of each condition to the user

table. The degree of match is calculated using the technique described previously in Section 5.8. In our evaluations we will also compare ProductServiceA against the user requests ProductRequest2, ProductRequest3 and ProductRequest4 and ProductServiceA completely matches all of these requests with a degree of match of 1.0. Note, ProductServiceA is the same as MovCin1 presented in Figure 6.2 in Section 6.3.1.

Class Concept		Weight	Class Definition
CinemaRequest	≡	(1.0)	∃ sellsProduct.MovieScreening ⊓
		(1.0)	∃ spatiallySubsumes.MovieCinema
InternetRequest	≡		∃ sellsProduct.(
		(1.0)	Internet ⊓
		(1.0)	∃ hasComponent.PCRequest)
PCRequest	≡	(1.0)	DesktopPC ⊓
		(0.9)	∃ hasComponent.SDReader ⊓
		(0.4)	∃ hasComponent.CDWriter
CafeRequest	≡		∃ sellsProduct.(
		(1.0)	∃ madeFrom.CoffeeBean ⊔
		(1.0)	∃ madeFrom.TeaLeaf)
PhotoRequest1	≡		∃ sellsProduct.(
		(1.0)	∃ output.{PhotoPaper10x15} ⊓
		(1.0)	∃ hasComponent.SDReader ⊓
		(1.0)	∃ duration.{Immediate})
PhotoRequest2	≡		∃ sellsProduct.(
		(1.0)	∃ output.{PhotoPaper10x15} ⊓
		(0.3)	∃ hasComponent.SDReader ⊓
		(0.1)	∃ duration.{Immediate})

Table 6.8: A listing of class definitions used by the user requests in Table 6.7 and explicit weights which express the importance of each condition to the user

Service	Match ProductRequest1	Description
ProductServiceA	Full Match 1.0	A movie cinema retail outlet with a cafe which has Internet available on desktop PCs with SD readers and CD writers and has immediate photo printing services with SD card reader (Note: ProductServiceA also fully matches ProductRequest 2, 3 and 4). ProductServiceA is the same as MovCin1 presented in Figure 6.2 in Section 6.3.1.
ProductServiceB	Partial Match 0.7	An cafe retail outlet which has Internet available on desktop PCs with SD readers and CD writers (no CinemaRequest)
ProductServiceC	Partial Match 0.63	A movie cinema retail outlet (no CafeRequest InternetRequest or PCRequest)
ProductServiceD	Partial Match 0.83	A movie cinema retail outlet with cafe which has a photo kiosk with a SD reader and CD writer (no Internet or DesktopPC)
ProductServiceE	Partial Match 0.88	A movie cinema retail outlet which has a cafe with WiFi Internet (no PCRequest)
ProductServiceF	Partial Match 0.86	A movie cinema retail outlet which has Internet access available on desktop PCs with SD readers and CD writers (no CafeRequest)
ProductServiceG	Partial Match 0.33	A retail outlet selling sweets (no CinemaRequest, InternetRequest, PCRequest or CafeRequest)
ProductServiceH	Failed Match 0.0	Car wash (no RetailOutlet CinemaRequest, InternetRequest, PCRequest, CafeRequest)

Table 6.9: A listing of eight service descriptions and their degree of match when compared to the weighted user request ProductRequest1

Printer Case Study

Bob is in a university campus and his mobile device has downloaded various service descriptions as he has moved around the campus. Bob wants to find a black and white, laser printer which supports wireless connectivity, which is also a fax machine and has some ink, so he can print some documents and send some faxes. However, different features / requirements have a different level of importance to Bob. This gives rise to PrinterRequest1 and PrinterRequest2 (which contain the same definitions as the non-weighted PrinterRequest1 from Section 6.4.1). These two requests contain the same requirements, but with different levels of importance associated with them.

In PrinterRequest1 the requirement for the printer to have ink (which implies it is operational) is important and the wireless requirement is not important because Bob does not mind also printing from a desktop PC. In PrinterRequest2 wireless support is very important to Bob since the files he needs to print are on his wireless device, but the fax and ink requirements are rated as less important. The requests, class definitions which they use, and the weight values associated with each conjunctive definition, which reflects the level of importance to Bob, are provided in Table 6.10. Note also, a role filler of a existential quantifier marked with curly parentheses (e.g. {Black} indicates an individual name Black, rather than a class type.

In our evaluations, we will compare the request PrinterRequest1 against 4 service descriptions which are given in Table 6.11. One service completely matches PrinterRequest1, 2 services partially match the user request and 1 service does not match any condition in PrinterRequest1. The degree to which each service matches PrinterRequest1 is also given in the table. The degree of match is calculated using the formula given previously in Section 5.8. Note, PrinterServiceA is the same as Printer1 presented in Figure 6.3 in Section 6.3.2

Class Concept		Weight	Class Definition
PrinterRequest1	≡	(1.0)	∃ LaserPrinterRequest ⊓
		(0.9)	∃ HasInkRequest ⊓
		(0.7)	∃ FaxRequest ⊓
		(0.3)	∃ BWRequest ⊓
		(0.1)	∃ WirelessRequest
PrinterRequest2	≡	(1.0)	∃ LaserPrinterRequest ⊓
		(0.75)	∃ WirelessRequest ⊓
		(0.5)	∃ BWRequest ⊓
		(0.25)	∃ FaxRequest ⊓
		(0.1)	∃ HasInkRequest
LaserPrinterRequest	≡	(1.0)	∃ input.Digital ⊓
		(1.0)	∃ output.Paper ⊓
		(0.2)	∃ hasCartridge.{Toner}
FaxRequest	≡	(1.0)	∃ supportsComm.(
		(1.0)	Fax ⊓
		(1.0)	≥ 1 hasPhoneNumber ⊓
		(1.0)	∃ hasPhoneNumber.Integer)
BWRequest	≡	(1.0)	∃ hasGraphicSupport.(
			∃ hasColourSupport.(
			∀ hasColour.{Black}))
WirelessRequest	≡	(1.0)	∃ supportsComm.Bluetooth ⊔
			∃ supportsComm.WiFi ⊔
			∃ supportsComm.IrDA
HasInkRequest	≡	(1.0)	∃ hasCartridge.(≥ 1 hasRemainingInk)

Table 6.10: A listing of class definitions representing two user requests, the class definitions these use, and explicit weights which express the importance of each condition to the user

Now that we have defined the weighted user requests and service descriptions for each of the product and printer case studies, we will use these in the next section to evaluate our adaptive inference strategy.

6.5.2 Mobile Performance Evaluation of the Adaptive Inference Strategy

In this section, we will present the degree of match obtained when comparing various weighted requests against various service descriptions, from the case

Service Name	Match Printer-Request1	Description
Printer-ServiceA	Full Match 1.0	A laser, fax, black and white, wireless, printer with a phone number and remaining ink notification. PrinterServiceA is the same as Printer1 presented in Figure 6.3 in Section 6.3.2
Printer-ServiceB	Partial Match 0.9	A laser, fax, black and white, printer with a phone number and remaining ink notification (no Wireless-Request)
Printer-ServiceC	Partial Match 0.76	A laser, black and white, wireless, printer with a phone number and remaining ink notification (no FaxRequest)
Printer-ServiceD	Failed Match 0.0	A phone box (no LaserPrinterRequest, FaxRequest, BWRequest, WirelessRequest or HasInkRequest)

Table 6.11: A listing of four service descriptions and their degree of match when compared to the weighted user request PrinterRequest1

studies detailed in the previous section. The results in this section compare our adaptive inference approach against standard inference / reasoning. In some evaluations the matching process is finished to completion and in others the reasoning process is interrupted before completion, either after a particular request requirement fails to match or after a time out period has elapsed. In this section, we evaluate the effectiveness of our adaptive inference strategy to:

- provide a degree of match result, rather than a binary *true* or $false$ result which standard reasoners provide;
- support adaptive and incremental reasoning which can be interrupted "anytime" based on user constrained such as available time or resources to provide a match result based on the computations completed, rather than the "all or nothing" principle that standard reasoners employ, which implies that all processing must be completed in order to obtain a result;

- support priority based reasoning where the most important request requirements / features, to the user, are checked first, rather than the arbitrary / depth-first ordering used by standard reasoners.

The results presented in this section are an average of two independent runs, since we performed the tests over two case studies comprising many service descriptions and requests.

In our first evaluation, we match the weighted request ProductRequest1, from the product case study against eight different service descriptions ProductServiceA - ProductServiceH, which were listed earlier in Table 6.7 in Section 6.5.1. We provide results using our adaptive inference strategy as well as standard reasoning. Two separate sets of results are provided for our adaptive strategy. In the first, we demonstrate that our adaptive strategy supports partial matching, by checking all conditions in the user request, even if some conditions fail (i.e. check all). In the second set of results we stop matching as soon as one request condition fails (i.e. stop on failure). The results are illustrated in Figure 6.15.

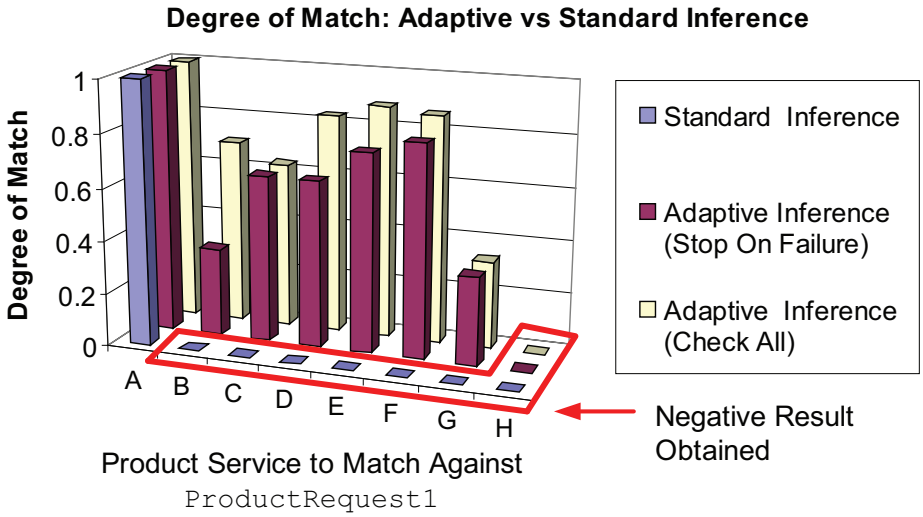


Figure 6.15: Comparison of degree of match for standard and adaptive reasoning, for matching of ProductRequest1 against ProductServiceA - ProductServiceH

The figure shows, in Test A, that standard reasoning successfully matched the service ProductServiceA with ProductRequest1. However, the standard reasoner gave a negative match result when comparing services ProductServiceB - ProductServiceG against ProductRequest1, in Tests B - G, even though these services partially matched the request. For instance, ProductServiceG is a movie cinema with Internet access, but does not serve tea / coffee. Therefore, ProductServiceG would still be of interest to the user, but a standard reasoner provides the same failed match result as for ProductServiceH which does not meet any of the requirements in the user request. Alternatively, when the adaptive inference was used compare all the requirements in the user request against the service description (check all), it provided the expected degree of match for all tests (see Table 6.9), and thus successfully supported partial matching. For instance, the comparison of ProductServiceF with ProductRequest1 gave a degree of match result of 0.86, which means that although ProductServiceF does not completely match the user request, it matched many of the conditions, and is thus probably useful to the user. When the adaptive strategy was stopped after a requested requirement failed to match (stop on failure) the potential service, the degree of match result was often less than the expected degree of match for all tests, but still meaningful to the user. In Tests C, G and H the results for stop on failure and check all were the same. This was because when a condition in the user request failed to match the service description, matching stopped for stop on failure, however, at that time all remaining conditions to check did not match the service description either.

In order to illustrate that the adaptive strategy works effective for other case studies, we now match the weighted request PrinterRequest1 from the printer case study against four different service descriptions PrinterServiceA - PrinterServiceD, which were listed earlier in Table 6.11 in Section 6.5.1. The results are illustrated in Figure 6.16.

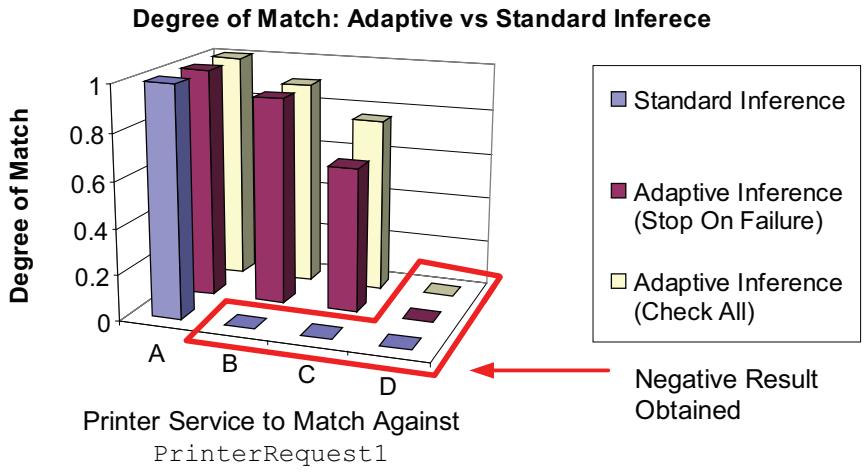


Figure 6.16: Comparison of degree of match for standard and adaptive reasoning, for matching of PrinterRequest1 against PrinterServiceA - PrinterServiceD

The figure shows, in Test A, that standard reasoning successfully matched the service PrinterServiceA with PrinterRequest1. However, the standard reasoner gave a negative match result when comparing services PrinterServiceB and PrinterServiceC against PrinterRequest1, in Tests B and C, even though these services partially matched the request. For instance, PrinterServiceB is a laser fax black and white printer with a mobile phone number and ink, but does not have wireless support. Therefore, PrinterServiceB would still be of interest to the user, but a standard reasoner provided the same result for PrinterServiceB as it provided for PrinterServiceD which does not meet any of the requirements in the user request. Alternatively, when the adaptive inference was used to check all the requirements in the user request (check all), it provided the expected degree of match for all Tests (see Table 6.11), and thus successfully supported partial matching. For instance, the comparison of PrinterServiceB with PrinterRequest1 gave a degree of match result of 0.9, which means that although ProductServiceB does not completely match the user request, it matched many of the conditions, and is thus probably useful to the user. When the adaptive strategy was stopped after a requested requirement failed to match the potential service (stop on failure), the degree of match

result was the same for Test B as when all request conditions were checked (check all), because the wireless condition (which did not match) was the last to be checked since it was of the least importance to the user. Conversely, in Test C a slightly reduced degree of match result was provided when using stop on failure compared to check all, but is still meaningful to the user.

The previous two evaluations clearly demonstrate that our adaptive reasoning strategy effectively provides an accurate and meaningful degree to which a particular service meets a user's request, based on the processing completed. This is much more useful to the user, than the standard reasoning approach which requires that all request conditions be successfully matched against the service description, otherwise a negative match result is provided to the user. These previous two evaluations also demonstrate the effect of interrupting the inference process after one request condition failed to match, showing the effectiveness of our strategy to support incremental reasoning. We will further investigate this feature in the next evaluations.

In the following evaluations we present the degree of match results which were obtained when the inference process was interrupted after 10, 20 and 30 seconds. First we will provide the results obtained for the product case study. Figure 6.17 presents the degree of match obtained when matching the request ProductRequest1 against the eight service descriptions ProductServiceA - ProductServiceH, which were previously listed in Table 6.7 in Section 6.5.1.

These eight comparisons are each shown in the figure as a separate test. As evidenced in the figure, a degree of match result can clearly be obtained after any period of processing time. When matching the request ProductRequest1 against many of the service advertisements, such as in Tests C, D, E, F, G, the greatest proportion of the degree of match result is obtained in the first 10 seconds of the matching process. This was because the most important request conditions (as deemed by the user) are checked first, and these contribute to a larger proportion of the degree of match result, than less significant conditions.

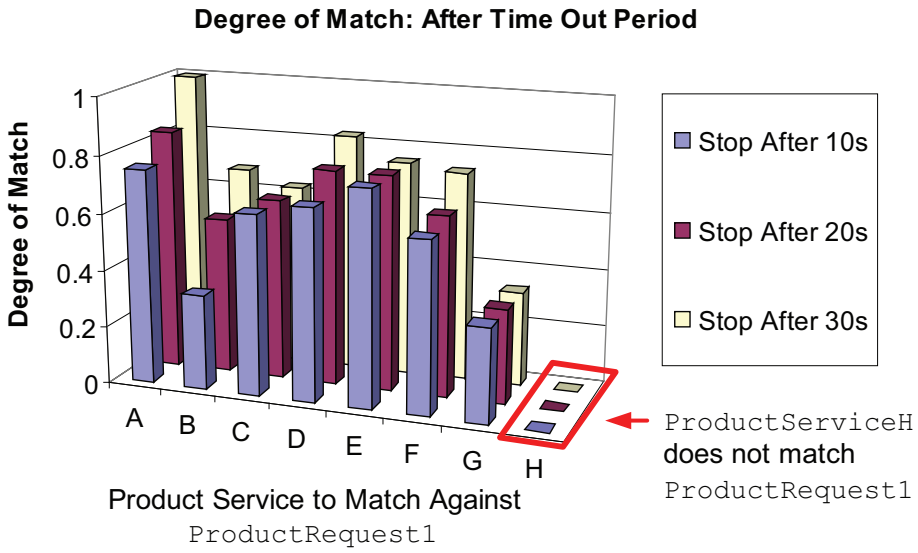


Figure 6.17: Comparison of degree of match for our adaptive inference strategy after a time out period of 10, 20 and 30 seconds, for the matching of ProductRequest1 against ProductServiceA - ProductServiceH

Alternatively, in some cases, such as Test B, main proportion of degree of match was not found until after 20 seconds of processing. This was because the important requirement for a movie cinema, was matched during the first 10 seconds, but failed to match, thereby not adding to the degree of match result. The ProductServiceH service description in Test H did not meet any requirement in the user request, so it did not matter when it was stopped.

We also performed the same tests using the printer case study. Figure 6.18 presents the degree of match obtained when matching the request PrinterRequest1 against the four service descriptions PrinterServiceA - PrinterServiceD, which were previously listed in Table 6.7 in Section 6.5.1.

When matching the request PrinterRequest1 against service descriptions PrinterServiceA, PrinterServiceB and PrinterServiceC, which was shown in Tests A, B and C, the greatest proportion of the degree of match result was obtained in the first 10 seconds, to differing extents. For Tests A, B and C, a degree of match of 0.6 was obtained in the first 10 seconds, because all three services matched the requirement for a printer which has ink, which was matched

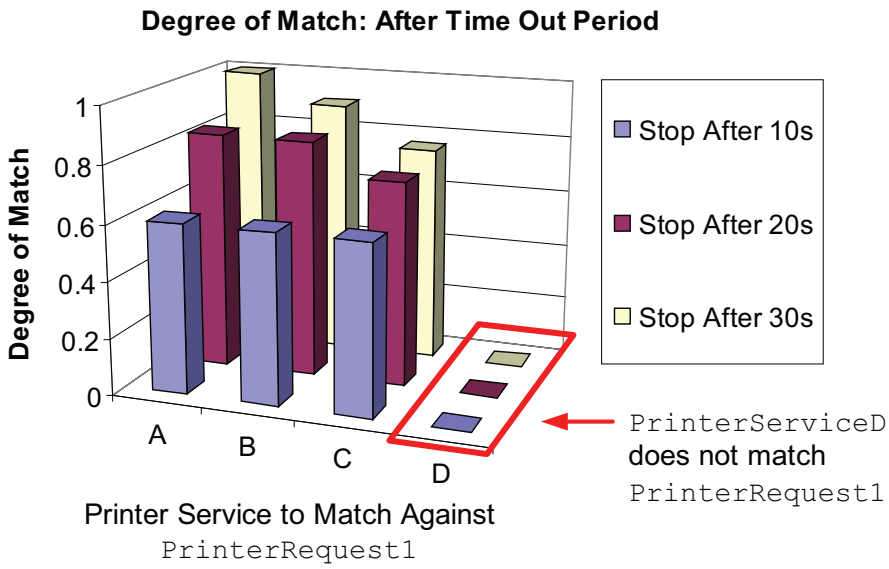


Figure 6.18: Comparison of degree of match for our adaptive inference strategy after a time out period of 10, 20 and 30 seconds, for the matching of PrinterRequest1 against PrinterServiceA - PrinterServiceD

first. The PrinterServiceB service description in Test B, did not match the wireless requirement which was checked in the time interval from 20-30 seconds the process. This lead to a reduced increase in the degree of match result when comparing the Test B which was stopped after 20 seconds to the Test B which was stopped after 30 seconds of processing. The PrinterServiceC service description, presented in Test C, did not match the fax requirement which was checked in the time interval from 10-20 seconds. This lead to a reduced increase in degree of match when comparing the Test C which was stopped after 10 seconds to the Test C which was stopped after 20 seconds of processing. However, other conditions (such as . The PrinterServiceD service in Test D did not meet any requirement, so it did not matter when it was stopped. This evaluation clearly demonstrates that our adaptive reasoning strategy effectively supports anytime reasoning, with a range of matching and partially matching service descriptions.

In the remainder of this section, we evaluate the priority matching employed by our adaptive inference strategy compared to standard reasoning. Priority

matching, means that the most important requirements in the user request are matched first. In this evaluation, we interrupt the reasoner after a time out period, and present the degree of match obtained when using our adaptive inference strategy and standard reasoning. When standard reasoning is utilised, the reasoner does not provide any result unless all conditions in the request were found to match the service description. Therefore, in the following evaluations, we will also provide a theoretical degree of match when using standard reasoning. The theoretical result is the degree of match which would be obtained based on the request conditions which have been successfully matched by the standard reasoner. It needs to be noted that this degree of match is computed based on the conditions / features in the user request evaluated by the standard reasoners at the time of the interruption. The standard reasoner chooses conditions in an arbitrary / depth-first order.

First we will present the results for the product case study. Figure 6.19 presents the degree of match obtained when matching ProductRequest1 against ProductServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

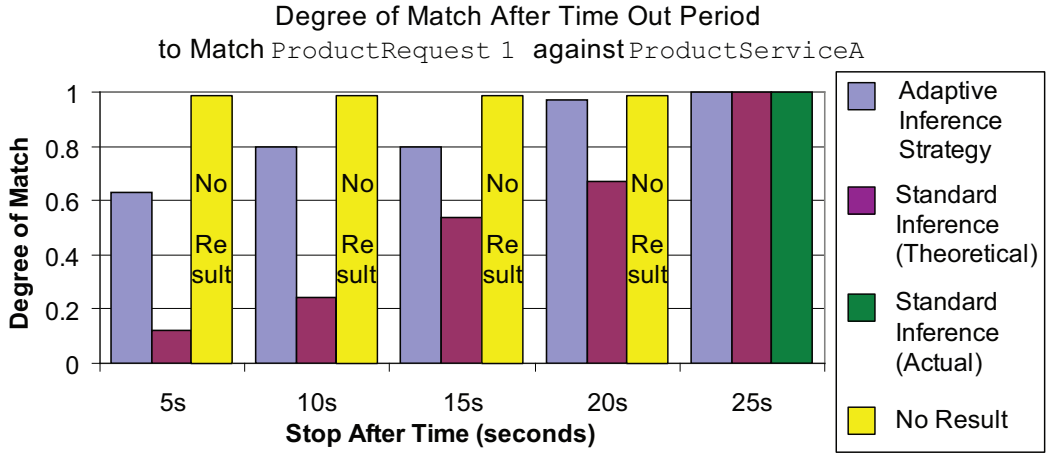


Figure 6.19: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 5 - 25 seconds, for matching ProductRequest1 against ProductServiceA

The figure presents the degree of match obtained in the case where the matching process is stopped after 5, 10, 15, 20 and 25 seconds, as separate

tests. As shown in the figure, after 5 seconds the adaptive reasoning strategy obtains a degree of match value of 0.63, because the requirement for a retail outlet which is a movie cinema, which were the most important requirements in the request, were checked first. The theoretical result for the standard reasoning approach is 0.12, because the requirement for a desktop PC with SD card reader and CD writer, was matched by the standard reasoning approach first. The order in which conditions in a request are matched by the standard reasoning strategy is arbitrary. However, standard reasoning actually returned no result when stopped after the first 5 seconds, because it did not finish matching all of the request. After 25 seconds the matching process was fully completed, meaning that all requirements in the request were checked. Since all requirements matched, the adaptive and standard reasoning strategies all return a degree of match value of 1.0 which denotes a full match. This evaluation shows that our adaptive inference strategy effectively employs priority matching, which matches the most important requirements in the request first. For instance, in the case that the reasoner is stopped after 10 seconds, our adaptive reasoning strategy returns a degree of match of 0.8. Conversely, the theoretical value for standard reasoning is 0.24, based on the request requirements matched in the 10 seconds of processing time. As such, the adaptive inference strategy effectively prioritises the matching time, to provide a much better indication of whether a service description matches the user request.

In Figure 6.20 we present the degree of match at at five second time intervals when comparing ProductRequest2 against ProductServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

The figure shows that after 5 seconds, the adaptive inference strategy obtained a degree of match of 0.61, because the important retail outlet and the cinema requirements was matched, while the theoretical result for standard reasoning was 0.05, because the less important SD card reader and CD writer

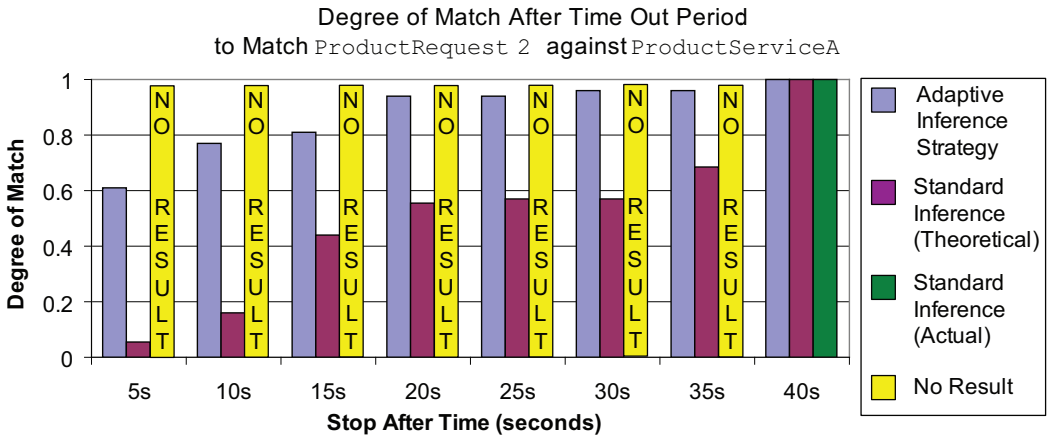


Figure 6.20: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 5 - 40 seconds, for matching ProductRequest2 against ProductServiceA

requirements were checked first. As the matching process continued the difference between the adaptive and standard results narrowed until the complete result of 1.0 was reached by both approaches was found after 40 seconds.

In Figure 6.21 we present the degree of match at at five second time intervals when comparing ProductRequest3 against ProductServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

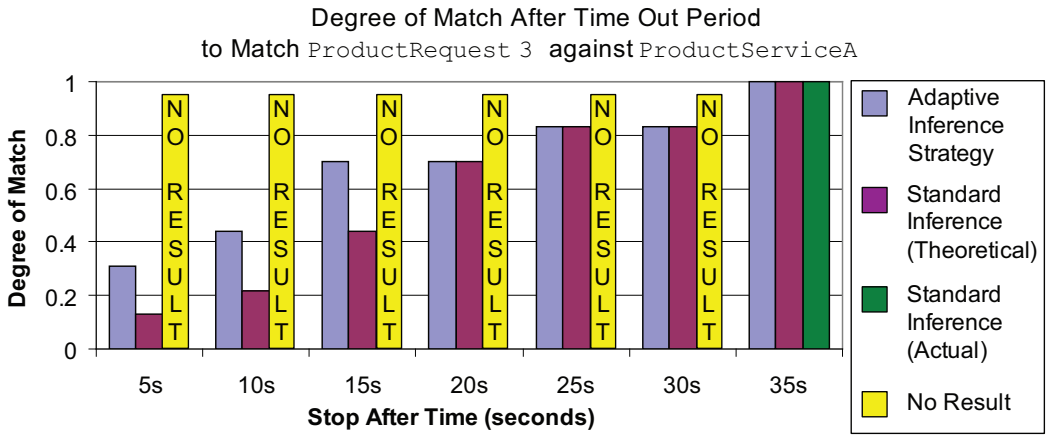


Figure 6.21: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 5 - 35 seconds, for matching ProductRequest3 against ProductServiceA

The figure shows that in the first three tests, which involve stopping the reasoner after 5, 10 and 15 seconds, the adaptive inference strategy successfully

matched the most important request requirements first and achieve a higher degree of match result and the theoretical standard reasoning result. In the remaining tests from 20 seconds and above, both the adaptive inference strategy and standard reasoner had matched the same conditions in the request leading to same degree of match for standard inference (theoretical) and adaptive inference. This was because after the first 15s when the remaining request requirements happened to be matched in the same order, coincidentally.

In Figure 6.22 we present the degree of match at at five second time intervals when comparing ProductRequest4 against ProductServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

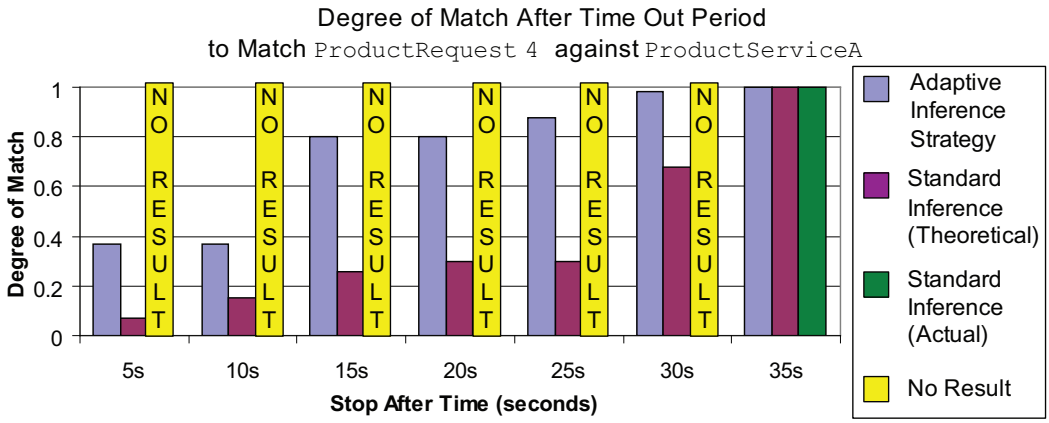


Figure 6.22: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 5 - 35 seconds, for matching ProductRequest4 against ProductServiceA

The figure shows after 5 seconds, the adaptive inference strategy obtained a degree of match of 0.37, because the important retail outlet and the cinema requirements was matched, while the theoretical result for standard reasoning was 0.07, because the less important SD card reader and CD writer requirements were checked first. As the matching process continued the difference between the adaptive and standard results narrowed until the complete result of 1.0 was reached by both approaches was found after 35 seconds.

In order to show that our adaptive inference strategy effectively employs priority matching across other scenarios, we also performed the same evaluation using the printer case study. In Figure 6.23 we present the degree of match at five second time intervals when comparing PrinterRequest1 against PrinterServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

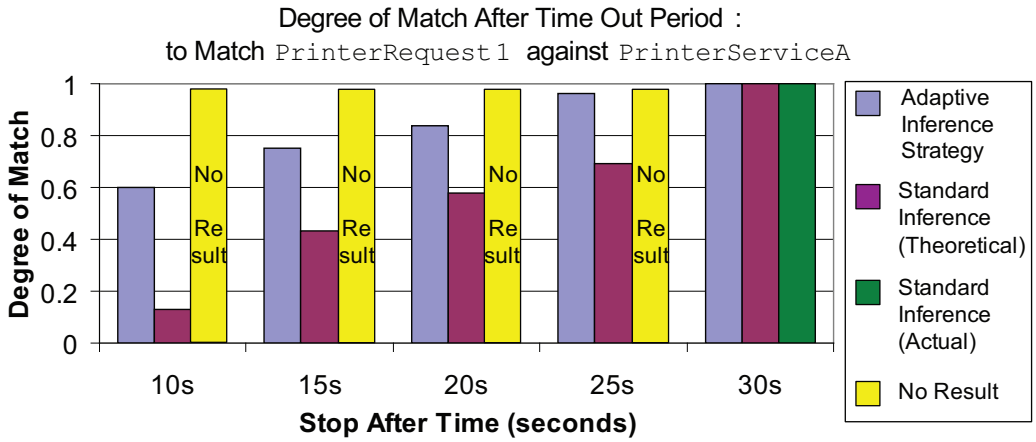


Figure 6.23: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 10 - 30 seconds, for matching PrinterRequest1 against PrinterServiceA

The figure shows after 10 seconds, the adaptive inference strategy obtained a degree of match of 0.6, because the important requirement for a printer which has ink was matched first, while the theoretical result for standard reasoning was 0.1, because the less important requirement for a black and white was checked first. As the matching process continued the difference between the adaptive and standard (theoretical) inference degree of match narrowed until the complete result of 1.0 was reached by both approaches was found after 30 seconds.

In Figure 6.24 we present the degree of match at five second time intervals when comparing PrinterRequest2 against PrinterServiceA, which fully matches the request. This request and service description were previously defined in Table 6.7 in Section 6.5.1.

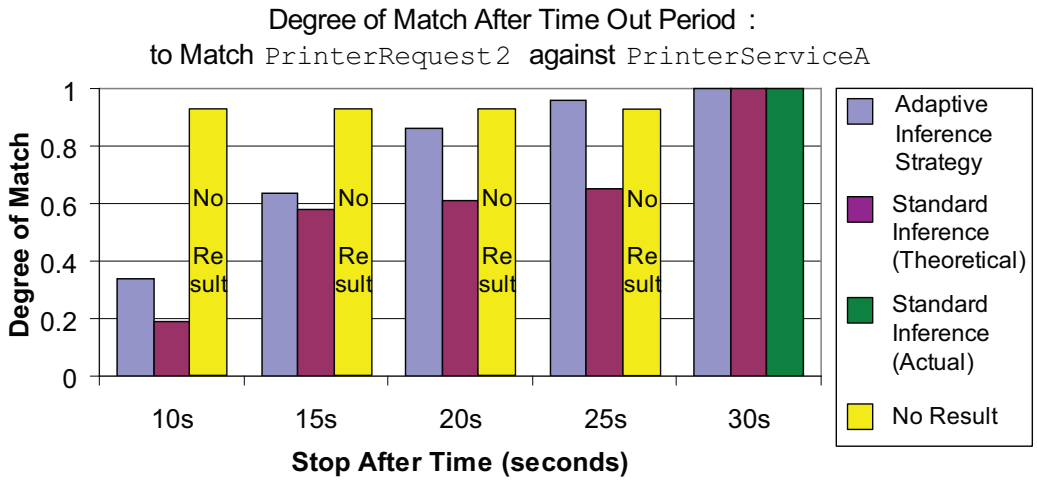


Figure 6.24: Comparison of degree of match for adaptive and standard reasoning after a time out period ranging from 10 - 30 seconds, for matching PrinterRequest2 against PrinterServiceA

The figure shows after 10 seconds, the adaptive inference strategy obtained a degree of match of 0.34, because the important requirement for a printer was matched first, while the theoretical result for standard reasoning was 0.19, because the less important requirement for a black and white was checked first. As the matching process continued the difference between the adaptive and standard (theoretical) inference degree of match results narrowed until the complete result of 1.0 was reached by both approaches was found after 30 seconds.

The previous six evaluations, show that our adaptive inference strategy effectively implements priority based matching and thus matches the most important request attributes first, as deemed by the user. If the matching process is completed in full, and all requested requirements are checked, then the order in which they are checked does not matter. This is why the theoretical standard inference and the adaptive inference degree of match results were identical when the matching task was completed in full. However, due to their inherent dynamic environment, mobile users are faced with time and resource constraints. This means that it may often be the case that there is not sufficient time or resources available to match a user request with a service description in full. In this situation, a partial result may be still useful to the user. This

result should make the best use of the time and resources available by matching the most important request requirements first. Our evaluations verify that our adaptive inference strategy effectively achieves this. Moreover, the evaluations presented in this section, show that when the matching process is interrupted prematurely, our adaptive inference strategy provides a higher degree of match and, therefore, a more useful and appropriate result to the user, than current reasoners, by implementing incremental, priority based, matching.

In this section, we have demonstrated that our adaptive inference strategy:

1. supports anytime matching and can be interrupted at any stage during the matching process. The matching process continues until there is not enough time or resources available to continue to provide a degree of match result to the user, based on the processing completed up until the time of the interruption. Conversely current reasoners are based on an "all or nothing" principle in which all conditions in the user request must be checked in order to provide a result;

2. supports priority based matching, meaning that the most important request conditions, as deemed by the user, are matched first. This ensures that if the matching process is interrupted early, a higher degree of match is returned to the user, than if request requirements were matched in an arbitrary order as with standard inference;

3. supports partial matching, where service descriptions do not completely match a user request. If a particular requirement in the user request fails, matching continues provided there is sufficient time and resources available to do so. Conversely current reasoners provide a failed match result in the case that any requirement in the user request fails to match, even if this condition was not important to the user.

6.6 Summary

In this chapter we presented the experimental evaluation of our light-weight mTableaux optimisation and caching strategies as well as our adaptive inference strategy. In our evaluation, we first compared mTableaux with other widely used reasoners in terms of response time and match result accuracy, on a desktop PC. This comparison used case studies which we developed as well as publicly available ontologies. We then provided a performance evaluation of mTableaux on small resource constrained mobile devices. This evaluation found that mTableaux successfully enabled mobile inference tasks on-board a resource constrained mobile device, to enable matching. It showed that using the optimisation strategies led to significant reductions in response times and that different strategies worked better for different case studies. However, using both optimisation strategies together achieved the best results in all case studies. The caching strategy produced further reductions in response time, where a request is partially or fully cached.

We also evaluated our adaptive inference strategy, for its effectiveness in adaptively supporting incremental, priority based matching which can be interrupted based on user constraints, such as time or resource availability, to provide a degree of match result based on the processing / computations completed at the time of interruption. We evaluated our strategy using a range weighted user requests which were compared against partially matching service descriptions, from two different case studies. The results show that our strategy can be effectively be interrupted at any stage during the reasoning process based on constraints such as limited time or resources and provides a degree of match result. The strategy was found to match the most important request requirements first. Therefore, our adaptive strategy produced a higher degree of match, than would be provided based on the request requirements matched by standard reasoning approaches, which matches requirements in an arbitrary order.

Having presented our theoretical contributions which included our lightweight mTableaux mobile inference strategies and our adaptive inference strategy, in Chapters 4 and 5, we have now completed the discussion of the contributions of this dissertation along with the evaluation in this chapter. In the following chapter we conclude this thesis.

Chapter 7

Conclusion

With the increasing focus on mobile applications and services, this thesis has examined, proposed and developed innovative techniques and applications to facilitate / enable the operation of semantic reasoners on mobile devices. A key challenge in enabling mobile semantic reasoning is the computational complexity of today's state-of-the-art semantic reasoners. In this context, this thesis has proposed, developed and experimentally evaluated strategies to meet this challenge of enabling mobile semantic reasoning without significantly reducing accuracy.

While there has been a substantial body of work in mobile and pervasive services, the potential role of semantic techniques for mobile environments is now being increasingly recognised by the emergence of research in this area (Kleemann and Sinner, 2006; Ruta et al., 2008a; Gu et al., 2007). Leveraging semantic reasoning to support service selection and matching has the known usefulness of improved accuracy and relevance to user requests. This dissertation makes contributions to this emerging area of mobile semantic reasoning.

7.1 Research Summary and Contributions

In this thesis we proposed strategies which focused on the aspects of mobile semantic reasoning with a view to improving the scalability and computational

performance of semantic reasoning techniques for mobile resource constrained devices. In this context, this thesis makes the following contributions:

- **Proposal and development of mTableaux optimisation and caching strategies to enable and improve the efficiency of on-board mobile reasoning**:

  Optimisation strategies: Due to the computational complexity of reasoning, current semantic reasoners cannot perform matching of large OWL-DL ontologies on small resource constrained mobile devices. Therefore, we proposed and developed two optimisation strategies which focus on improving performance efficiency by relaxing the strict adherence to completeness. While accuracy is important, these strategies also consider it equally necessary that information by provided with improved response time. In our evaluation we showed that our strategies provided a significant performance improvement compared to current commercial and open source reasoners while maintaining a high degree of accuracy. In addition, an evaluation on a resource constrained mobile device demonstrated that using both strategies together provides significant improvements to performance on such a device.

  Caching strategy: Many current semantic reasoners cache the results of a matching task. However, current semantic reasoners do not cache the evaluations performed as part of the matching process, so that they can be subsequently used for similar matching tasks. Therefore, given the growth of secondary storage media, we incorporate caching mechanisms to semantic matching tasks in order to improve response time. We proposed a caching strategy which stores the results of previous requests so that they can be used in similar future requests. Our strategy supports two main modes of operation where:

  1. a cached entry is used in place of the matching process or

2. a cached entry is used to priorities the application of parts of the matching process led to a successful match result previously, but require the re-evaluation of this process in case the service description has changed.

Our evaluations showed that our caching strategy led to significant performance improvements in both modes of operation;

- **Proposal and development of a resource adaptive priority inference strategy**. Although many current service matching approaches support partial matching of user requests against service descriptions, current semantic reasoners do not. Rather, current reasoners provide only a positive or negative result and under an "all or nothing" principle which requires that the matching process be completed in full otherwise no result is provided. Therefore, our proposed adaptive inference strategy supports incremental matching of the requirements in the user request against a service description. This process can be interrupted prematurely and intermediate results provided to the user. Typical situations or constraints that may necessitate such intermediate results in a mobile environment are, the user's need for immediate results (i.e. lack of time available), changing user context (e.g. location), as well as device context (e.g. low battery levels, inadequate memory levels). This strategy supports the association of a level of importance to each requirement in the user request and matches requirements in importance order. A weighted degree of match is provided to the user based on the computations / inference checks performed up to the point at which the matching process ends. We performed an experimental evaluation comparing our adaptive strategy with standard approaches on a resource constrained device under certain time constraints. Our evaluation demonstrated that our approach provided an equivalent or greater degree of match result

than standard reasoning approaches at various points of interruption and effectively supported incremental matching.

The above discussion has highlighted the principal contributions of this dissertation. In the next section we will briefly discuss future directions of this work.

7.2 Future Research Directions

The principle contributions of this research were in the semantic reasoner module of a service matching architecture. There are several possible future research directions which could extend from our approach:

- In our adaptive inference strategy, presented in Chapter 5, the matching process may be interrupted prematurely due to limits of time, resources or other user constraints. We evaluated this strategy in Chapter 6 by using time as a constraint. This evaluation could be expanded to include other constraints such as available memory, battery life, etc. In addition, the decision about whether to interrupt the matching process or continue matching could include other factors, such as the current intermediate degree of match, based on those conditions in the user's request which have already been compared against the current service description. For instance, since our strategy is priority ordered, the most important requirement in the user request will be checked first. If important conditions fail to match the current service description, it may be a better use of constrained time or resources to begin checking another service description.

- The focus of our contributions were to enable mobile semantic inference proof. In Chapter 3 we illustrated a generic architecture which may use our mobile inference proof approach for service matching. In this

dissertation we did not specify how the additional modules of this architecture should be implemented. A future research direction could be to implement the other modules of this architecture, such as the context-aware realisation module, matching engine and advertisement database / repository.

- Constructing user preferences based on the historical usage patterns of the user and using these preferences for pre-emptive processing could be another extension of this research. For instance, if the user has previously searched for sales of baby clothing in shopping centres, the architecture may begin narrowing down service descriptions about baby clothes on sale as soon as the user enters the car park of a shopping centre next time. Such pre-emptive processing would thus be conducted when the user is not using the device, in order to speed up the matching process at request-time. Additionally, the user may wish to specify explicit preferences using rules. These rules could be used to pre-select services from the list of potential services which are each matched against the user request. For instance, the user may specify a preference for services which are closer to him or her. In this case, the services which are located close to the user should be checked first.

- In our architecture we loaded all service descriptions, which have been downloaded by the user's device, into main memory. This occured before the matching process begun. In some cases ontologies containing service descriptions may be extremely large. These ontologies may be too large for a resource constrained device to load completely into memory. The generic architecture for service selection, presented in Chapter 3, could be extended to utilise strategies for partitioning ontologies into smaller subsets such as Stuckenschmidt and Klein (2004); Stuckenschmidt and Schlicht (2009); Grau et al. (2009); Guo and Heflin (2006). Each subset

may contain similar definitions. Each ontology subset could be loaded into the reasoner only as required by the matching process.

In conclusion, this thesis takes a step forward in realising the potential of mobile semantic reasoning and enabling semantic services in a mobile environment. The contributions of this research and the possibilities created for future development have demonstrated the potential of using semantic based service matching in a mobile setting.

References

Abramowicz, W., Haniewicz, K., Kaczmarek, M. and Zyskowski, D. (2008). E-marketplace for semantic web services, *6th International Conference on Service-Oriented Computing (ICSOC '08)*, Vol. 5364, Springer-Verlag, Sydney, Australia, pp. 271 – 285.

Agostini, A., Bettini, C. and Riboni, D. (2007). A performance evaluation of ontology-based context reasoning, *5th Pervasive Computing and Communications Workshops, in conjunction with the International Conference on Pervasive Computing (PerCom '07)*, IEEE, White Plains, NY, USA, pp. 3 – 8.

Aijaz, F., Ali, S., Chaudhary, M. and Walke, B. (2009). Enabling high performance mobile web services provisioning, *70th Vehicular Technology Conference (VTC '09)*, IEEE, Anchorage, AK, USA, pp. 1 – 6.

Ajtai, M.; Gurevich, Y. (1989). Datalog vs. first-order logic, *30th Annual Symposium on Foundations of Computer Science*, IEEE, Research Triangle Park, NC, USA, pp. 142 – 147.

Almeida, D. R. d., Bapista, C. d. S., Silva, E. R. d., Campelo, C. E. C., Figueiredo, H. F. d. and Lacerda, Y. A. (2006). A context-aware system based on service-oriented architecture, *20th International Conference on Advanced Information Networking and Applications(AINA '06)*, IEEE, Vienna, Austria, pp. 205 – 210.

Arnold, K., O'Sullivan, B., Scheifler, R. W., Waldo, J. and Woolrath, A. (1999). *The Jini Specification*, 2nd edn, ISBN: 0201726173, Addison-Wesley.

Asif, M., Majumdar, S. and Dragnea, R. (2007). Hosting web services on resource constrained devices, *International Conference on Web Services (ICWS '07)*, IEEE, Salt Lake City, UT, pp. 583 – 590.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. and Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*, ISBN: 0521781760, Cambridge University Press.

Baader, F. and Hanschke, P. (1991). A scheme for integrating concrete domains into concept languages, *12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, Morgan Kaufmann, Sydney, Australia, pp. 452 – 457.

Baader, F., Horrocks, I. and Sattler, U. (2005). Description logics as ontology languages for the semantic web, *Mechanizing Mathematical Reasoning*, Vol. 2605, Springer-Verlag, pp. 228 – 248.

Baader, F. and Sattler, U. (2001). An overview of tableau algorithms for description logics, *Studia Logica* **69**(1): 5 – 40.

Baousis, V., Spiliopoulos, V., Zavitsanos, E., Hadjiefthymiades, S. and Merakos, L. (2008). Semantic web services and mobile agents integration for efficient mobile services, *International Journal on Semantic Web and Information Systems (IJSWIS), IGI* **4**(1): 1 – 19.

Bechhofer, S., Moller, R. and Crowther, P. (2003). The DIG description logic interface, *International Workshop on Description Logics (DL '03)*, CEUR-WS.org.

URL: *http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-81/bechhofer.ps (accessed March 2010)*

Bener, A. B., Ozadali, V. and Ilhan, E. S. (2009). Semantic matchmaker with precondition and effect matching using SWRL, *Expert Systems with Applications, Pergamon Press* **36**(5): 9371 – 9377.

Bernstein, A. and Klein, M. (2002). Discovering services: Towards high-precision service retrieval, *International Workshop on Web Services, E-Business, and the Semantic Web (CAiSE '02)*, Vol. 2512, Springer-Verlag, Toronto, Canada, pp. 260 – 275.

Bianchini, D., De Antonellis, V., Melchiori, M. and Salvi, D. (2006). Lightweight ontology-based service discovery in mobile environments, *17th International Conference on Database and Expert Systems Applications (DEXA '06)*, IEEE, Krakow, Poland, pp. 359 – 364.

Billinghurst, M. and Starner, T. (1999). Wearable devices: New ways to manage information, *Computer, IEEE* **32**(1): 57 – 64.

Blefari-Melazzi, N., Casalicchio, E. and Salsano, S. (2007). Context-aware service discovery in mobile heterogeneous environments, *16th IST Mobile and Wireless Communications Summit*, IEEE, Budapest, Hungary, pp. 1 – 5.

Borgida, A. and Patel-Schneider, P. F. (1994). A semantics and complete algorithm for subsumption in the classic description logic, *Journal of Artificial Intelligence Research* **1**: 277 – 308.

Bouillet, E., Feblowitz, M., Liu, Z., Ranganathan, A. and Riabov, A. (2008). Semantic models for ad hoc interactions in mobile, ubiquitous environments, *International Conference on Semantic Computing*, IEEE, Santa Clara, CA, pp. 589 – 596.

Brambilla, M., Celino, I., Ceri, S., Cerizza, D., della Valle, E., Facca, F. and Tziviskou, C. (2006). Improvements and future perspectives on web engineering methods for automating web services mediation, choreography

and discovery: SWS-challenge phase III, *3rd Workshop of the Semantic Web Service Challenge*, Athens, GA, USA.

URL: *http://sws-challenge.org/workshops/2006-Athens/papers/SWS-phase-III_polimi_cefriel_v1.0.pdf (accessed Feb 2010)*

Broens, T., Pokraev, S., Sinderen, M. v., Koolwaaij, J. and Dockhorn Costa, P. (2004). Context-aware, ontology based, service discovery, *European Symposium on Ambient Intelligence (EUSAI'04)*, Vol. 3295, Springer-Verlag, Eindhoven, the Netherlands, pp. 72 – 83.

Buchanan, M. (2009). Mobile revolution: it's the year of the smartphone.

URL: *http://www.smh.com.au/news/national/mobile-revolution-its-the-year-of-the-smartphone/2009/01/03/1230681809474.html (accessed May 2009)*

Buchheit, M., Donini, F. M. and Schaerf, A. (1993). Decidable reasoning in terminological knowledge representation systems, *13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, Morgan Kaufmann, Los Altos, pp. 704 – 709.

Calvanese, D., De Giacomo, G., Lenzerini, M. and Nardi, D. (2001). Reasoning in expressive description logics, *Handbook of Automated Reasoning, Elsevier* **2**: 1581 – 1634.

Campo, C., Garcia-Rubioa, C., Lopeza, A. M. and Almenareza, F. (2006). PDP: A lightweight discovery protocol for local-scope interactions in wireless ad-hoc networks, *Computer Networks, Elsevier* **50**(17): 3264 – 3283.

Ceri, S., Gottlob, G. and Tanca, L. (1989). What you always wanted to know about datalog (and never dared to ask), *Transactions on Knowledge and Data Engineering, IEEE* **1**: 146 – 166.

Chakraborty, D., Joshi, A., Yesha, Y. and Finin, T. (2006). Towards distributed service discovery in pervasive computing environments, *Transactions on Mobile Computing, IEEE* **5**(1): 97 – 112.

Chakraborty, D., Perich, F., Avancha, S. and Joshi, A. (2001). Dreggie: Semantic service discovery for m-commerce applications, *Workshop on Reliable and Secure Applications in Mobile Environment in Conjunction with 20th Symposium on Reliable Distributed Systems (SRDS)*, New Orleans, Louisiana, USA.

URL: *http://ebiquity.umbc.edu/paper/html/id/49/ (accessed March 2010)*

Chatti, M. A., Srirama, S., Kensche, D. and Cao, Y. (2006). Mobile web services for collaborative learning, *4th International Workshop on Wireless, Mobile and Ubiquitous Technology in Education*, IEEE, Athens, Greece, pp. 129 – 133.

Chen, H., Finin, T. and Joshi, A. (2004). Semantic web in the context broker architecture, *2nd International Conference on Pervasive Computer and Communications (PerCom 04)*, IEEE, Orlando, Florida, pp. 277 – 286.

Chen, H., Finin, T. and Joshi, A. (2005). The SOUPA ontology for pervasive computing, *Ontologies for Agents: Theory and Experiences, Birkhauser Basel* pp. 233 – 258. Springer-Verlag.

Chen, I., Yang, S. and Zhang, J. (2006). Ubiquitous provision of context aware web services, *International Conference on Services Computing (SCC '06)*, IEEE, Chicago, IL, pp. 60 – 68.

Choi, K.-H., Lee, K. M., Shin, H. J. and Shin, D.-R. (2005). Efficient algorithm for service matchmaking in ubiquitous environments, *6th International Conference on E-Commerce and Web Technologies (EC-Web '05)*, Vol. 3590, Springer-Verlag, Copenhagen, Denmark, pp. 258 – 266.

Church, A. (1936). A note on the entscheidungs problem, *Journal of Symbolic Logic* **1**: 40 – 41.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms*, 2nd edn, ISBN: 0262032937, MIT Press.

de Andrade, F., Schiel, U. and de Souza Baptista, C. (2007). Constraint-based web services discovery and composition, *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM '07)*, IEEE, Papeete, France, pp. 118 – 124.

de Freitas, S. and Levene, M. (2003). Wearable technology: Evaluating wearable devices, PDAs & other mobile devices in FE & HE institutions, Joint Information Systems Committee (JISC) Technology and Standards Watch Report.
URL: *http://www.jisc.ac.uk/whatwedo/services/techwatch/reports/-horizonscanning/hs0305.aspx (accessed November 2009)*

De, S. and Moessner, K. (2008). Ontology-based context inference and query for mobile devices, *19th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '08)*, IEEE, Dalian, China, pp. 1 – 5.

desJardins, M., Eaton, E. and Wagstaff, K. L. (2006). Learning user preferences for sets of objects, *23rd International Conference on Machine Learning (ICML '06)*, ACM, Pittsburgh, Pennsylvania, pp. 273 – 280.

Dietze, S., Gugliotta, A. and Domingue, J. (2009). Exploiting metrics for similarity-based semantic web service discovery, *7th International Conference on Web Services (ICWS '09)*, IEEE, Los Angeles, CA, USA, pp. 327 – 334.

Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B. and Pedrinaci, C. (2008). IRS-III: A broker-based approach to semantic web services, *Journal of Web Semantics, Elsevier* **6**(2): 109 –132.

Doulkeridis, C. and Vazirgiannis, M. (2008). CASD: Management of a context-aware service directory., *Pervasive and Mobile Computing, Elsevier* **4**(5): 737 – 754.

Doyle, J. and Patil, R. S. (1991). Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services, *Artificial Intelligence, Elsevier* **48**: 261 – 297.

Eiter, T., Gottlob, G. and Mannila, H. (1997). Disjunctive datalog, *Transactions on Database Systems (TODS), ACM* **22**: 364 – 418.

El-Sayed, A.-R. and Black, J. P. (2006). Semantic-based context-aware service discovery in pervasive-computing environments, *1st International Workshop on Services Integration in Pervasive Environments in-conjunction with the International Conference on Pervasive Services (ICPS '06)*, Lyon, France, pp. 7 – 12.

Fensel, D. and Bussler, C. (2002). The web service modeling framework WSMF, *Electronic Commerce Research and Applications, Elsevier* **1**(2): 113 – 137.

Fensel, D. and van Harmelen, F. (2007). Unifying reasoning and search to web scale, *Internet Computing, IEEE* **11**(2): 96, 94 – 95.

Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Valle, E. D., Fischer, F., Huang, Z., Kiryakov, A., il Lee, T. K., Schooler, L., Tresp, V., Wesner, S., Witbrock, M. and Zhong, N. (2008). Towards LarKC: A platform for web-scale reasoning, *2nd International Conference on Semantic Computing (ICSC '08)*, IEEE, Santa Clara, CA, USA, pp. 524 – 529.

Ferscha, A. (2009). *Hagenberg Research, Chapter VIII Pervasive Computing*, ISBN: 978-3-642-02126-8, Springer-Verlag.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*, 2 edn, ISBN: 0387945938, Springer-Verlag.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem, *Artificial Intelligence* **19**: 17 – 37.

Galton, A. (1990). *Logic for Information Technology*, ISBN: 0471927775, Wiley.

Gartner (2008). Worldwide smartphone sales grew 29 percent in first quarter of 2008, Gartner Press Releases.
URL: *http://www.gartner.com/it/page.jsp?id=754112 (accessed February 2010)*

Gehlen, G. and Pham, L. (2005). Mobile web services for peer-to-peer applications, *2nd Consumer Communications and Networking Conference (CCNC '05)*, IEEE, Las Vegas, Nevada, USA, pp. 427 – 433.

Girle, R. (2001). *Modal Logics and Philosophy*, 2nd edn, ISBN: 0773521496, Teddington UK, McGill-Queen's University Press.

Giugno, R. and Lukasiwicz, T. (2002). P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web, *European Conference on Logics in Artificial Intelligence (JELIA '02)*, Vol. 2424, Springer-Verlag, Cosenza, Italy, pp. 86 – 97.

Gonzalez-Castillo, J., Trastour, D. and Bartonlini, C. (2001). Description logics for matchmaking of services, *Workshop on Applications of Description Logics (KI '01)*, CEUR-WS.org, Vienna, Austria.
URL: *http://www.hpl.hp.com/techreports/2001/HPL-2001-265.pdf (accessed March 2010)*

Gore, R. (1998). *Handbook of Tableau Methods*, ISBN: 9780792356271, Kluwer, Dordrecht.

Grau, B. C., Horrocks, I., Kazakov, Y. and Sattler, U. (2009). Extracting modules from ontologies: A logic-based approach, *Modular Ontologies, Springer-Verlag* **544**: 159 – 186.

Grosof, B. N., Horrocks, I., Volz, R. and Decker, S. (2003). Description logic programs: Combining logic programs with description logic, *12th International Conference on World Wide Web (WWW '03)*, ACM, Budapest, Hungary, pp. 48 – 57.

Gu, T., Kwok, Z., Koh, K. K. and Pung, H. K. (2007). A mobile framework supporting ontology processing and reasoning, *2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructure (RSPS) in conjunction with the International Conference on Ubiquitous Computing (Ubicomp '07)*, Innsbruck, Austria.
URL: *http://www.igd.fhg.de/igd-a1/RSPSI2/papers/-Ubicomp2007_RSPSI2_TaoGu.pdf (accessed March 2010)*

Guo, Y. and Heflin, J. (2006). A scalable approach for partitioning owl knowledge bases, *International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS '06) in conjunction with 5th International Semantic Web Conference (ISWC '06)*, Athens, GA, USA, pp. 47 – 60.

Guttman, E. (1999). Service location protocol: Automatic discovery of IP network services, *Internet Computing, IEEE* **3**(4): 71 – 80.

Han, S., Zhang, S. and Zhang, Y. (2006). Dynamic selection and optimization of design paradigms in mobile services, *Asia-Pacific Conference on Services Computing (APSCC '06)*, Guangzhou, Guangdong, pp. 646 – 649.

Hartman, F. and Reynolds, H. (2004). Was the universal service registry a dream?, SOA World Magazine.
URL: *http://soa.sys-con.com/node/47278 (accessed March 2010)*

Heinsohn, J., Kudenko, D., Nebel, B. and Profitlich, H.-J. (1994). An empirical analysis of terminological representation systems, *Artificial Intelligence* **68**(2): 367 – 397.

Helal, S. (2002). Standards for service discovery and delivery, *Pervasive Computing, IEEE* **1**(3): 95 – 100.

Hepp, M. (2006). Products and services ontologies: A methodology for deriving OWL ontologies from industrial categorization standards, *International Journal on Semantic Web and Information Systems (IJSWIS), IGI* **2**(1): 72 – 99.

Hitzler, P. and Vrandecic, D. (2005). Resolution-based approximate reasoning for OWL DL, *International Semantic Web Conference (ISWC '05)*, Springer-Verlag, Galway, Ireland, pp. 383 – 397.

Hliaoutakis, A., Varelas, G., Voutsakis, E., Petrakis, E. G. M. and Milios, E. (2006). Information retrieval by semantic similarity, *International Journal on Semantic Web and Information Systems (IJSWIS), IGI* **2**(3): 55 – 73.

Hollunder, B., Nutt, W. and Schmidt-SchauSS., M. (1990). Subsumption algorithms for concept description languages, *9th European Conference on Artificial Intelligence (ECAI '90)*, Pitman, London, UK, pp. 348 – 353.

Horn, A. (1951). On sentences which are true of direct unions of algebras, *Journal of Symbolic Logic* **16**: 14 – 21.

Horrocks, I. and Patel-Schneider, P. F. (1999). Optimising description logic subsumption, *Journal of Logic and Computation* **9**(3): 267 – 293.

Horrocks, I. and Sattler, U. (2007). A tableaux decision procedure for SHOIQ, *Journal of Automated Reasoning, Springer-Verlag* **39**(3): 249 – 276.

Howes, T. A. and Smith, M. C. (1995). A scalable, deployable directory service framework for the internet, *Technical Report 95-7*, University of Michigan. **URL:** *http://www.citi.umich.edu/techreports/reports/citi-tr-95-7.pdf (accessed March 2010)*

Hustadt, U., Motik, B. and Sattler, U. (2004). Reducing SHIQ description logic to disjunctive datalog programs, *9th International Conference on Knowledge Representation and Reasoning (KR '04)*, Whistler, Canada, pp. 152 – 162.

Hustadt, U., Motik, B. and Sattler, U. (2007). Reasoning in description logics by a reduction to disjunctive datalog, *Journal of Automated Reasoning, Springer-Verlag* **39**(3): 351 –384.

Iranmanesh, Z., Piri, R. and Abolhassani, H. (2009). An approach for semantic web query approximation based on domain knowledge and user preferences, *Advances in Computer Science and Engineering: Selected Papers from 13th International CSI Computer Conference (CSICC '08)*, Vol. 6 of *Communications in Computer and Information Science*, Springer-Verlag, Kish Island, Iran, pp. 443 – 452.

Islam, N. and Shaikh, Z. (2008). A novel approach to service discovery in mobile adhoc network, *International Networking and Communications Conference (INCC 08)*, IEEE, Lahore Pakistan, pp. 58 – 62.

Jeon, H., Kim, T. and Choi, J. (2008). Mobile semantic search using personal preference filtering, *4th International Conference on Networked Computing and Advanced Information Management (NCM '08)*, IEEE, Gyeongju, South Korea, pp. 531 – 534.

Jiang, J. J. and Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy, *International Conference on Research in Computational Linguistics*, pp. 19 – 33.

Jung, K.-Y. (2006). User preference through learning user profile for ubiquitous recommendation systems, *10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES '06)*, Vol. 4251, Springer-Verlag, Bournemouth, UK, pp. 163 – 170.

Kargin, B. and Basoglu, N. (2007). Factors affecting the adoption of mobile services, *Portland International Center for Management of Engineering and Technology*, IEEE, Portland, OR, USA, pp. 2993 – 3001.

Kelly, J. J. (1997). *The Essence of Logic*, 1st edn, ISBN: 0133963756, Prentice-Hall.

Kifer, M., Lausen, G. and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages, *Journal of the Association for Computing Machinery, ACM* **42**(4): 741 – 843.

Kim, Y.-S. and Lee, K.-H. (2007). A light-weight framework for hosting web services on mobile devices, *5th European Conference on Web Services (ECOWS '07)*, IEEE, Halle, Germany, pp. 255 – 263.

Kleemann, T. (2006). Towards mobile reasoning, *International Workshop on Description Logics (DL '06)*, Windermere, Lake District, UK.
URL: *http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-189/submission_36.pdf (accessed March 2010)*

Kleemann, T. and Sinner, A. (2006). User profiles and matchmaking on mobile phones, *16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP '06)*, Vol. 4369, Springer-Verlag, Fukuoka, Japan, pp. 135 – 147.

Konar, A. (2005). *Computational Intelligence: Principles, Techniques and Applications*, 1st edn, ISBN: 3540208984, Springer-Verlag.

Koodli, R. and Perkins, C. (2002). Service discovery in mobile on-demand ad hoc networks, IETF Internet Draft.
URL: *http://people.nokia.net/ rajeev/manetsrv.txt (accessed March 2010)*

Kurkovsky, S., Zanev, V. and Kurkovsky, A. (2005). SMMART, a context-aware mobile marketing application: Experiences and lessons, *Embedded and Ubiquitous Computing*, Vol. 3823, Springer-Verlag, Nagasaki, Japan, pp. 141 – 150.

Kuster, U. and König-Ries, B. (2008). Semantic service discovery with DIANE service descriptions, *Semantic Web Services Challenge*, Vol. 8, Springer-Verlag, pp. 199 – 216.

Lee, C., Helal, A., Desai, N., Verma, V. and Arslan, B. (2003). Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services, *Transactions on Systems, Man and Cybernetics, IEEE* **33**(6): 682 – 696.

Li, L. and Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology, *International Journal of Electronic Commerce* **8**(4): 39 – 60.

Li, L. and Lamont, L. (2005). A lightweight service discovery mechanism for mobile ad-hoc pervasive environment using cross-layer design, *3rd International Conference on Pervasive Computing and Communications (Percom '05) Workshops*, IEEE, Kauai Island, Hawaii, pp. 55 – 59.

Li, L., Liu, Q., Tao, Y., Zhang, L., Zhau, J. and Yu, Y. (2006). Providing an uncertainty reasoning service for semantic web application, *Frontiers of WWW Research and Development (APWeb '06)*, Springer-Verlag, Harbin, China, pp. 628–639.

Li, Z., Wang, C. and Xu, R. (2001). Computation offloading to save energy on handheld devices: A partition scheme, *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ACM, Atlanta, Georgia, USA, pp. 238 – 246.

Liang, J.-C., Chen, J.-C. and Zhang, T. (2007). Mobile service discovery protocol (MSDP) for mobile ad-hoc networks, *8th International Symposium on Autonomous Decentralized Systems (ISADS '07)*, IEEE, Sedona, AZ, USA, pp. 352 – 362.

Lin, D. (1998). An information-theoretic definition of similarity, *International Conference on Machine Learning (ICML '98)*, Morgan Kaufmann, San Francisco, CA, USA, pp. 296 – 304.

Lloyd, J. W. (1987). *Foundations of Logic Programming*, 2nd edn, ISBN: 3-540-18199-7, Springer-Verlag.

Lu, H. (2005). Semantic web services discovery and ranking, *International Conference on Web Intelligence*, IEEE, Los Alamitos, CA, USA, pp. 157 – 160.

Lukasiewicza, T. and Straccia, U. (2009). Description logic programs under probabilistic uncertainty and fuzzy vagueness, *International Journal of Approximate Reasoning, Elsevier* **50**(6): 837 – 853.

Luo, J., Montrose, B. and Kang, M. (2005). An approach for semantic query processing with UDDI, *On the Move to Meaningful Internet Systems*, Vol. 3762, Springer-Verlag, pp. 89 – 98.

Luo, J., Montrose, B., Kim, A., Khashnobish, A. and Kang, M. (2006). Adding OWL-S support to the existing UDDI infrastructure, *International Conference on Web Services (ICWS '06)*, IEEE, Chicago, IL, USA, pp. 153 – 162.

MacGregor, R. (1991). Inside the LOOM description classifier, *Special Issue on Implemented Knowledge Representation and Reasoning Systems (SIGART) Bulletin, ACM* **2**(3): 88 – 92.

Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E. and Srinivasan, N. (2007). Bringing semantics to web services with OWL-S, *World Wide Web, Springer-Verlag* **10**(3): 243 – 277.

Masuoka, R., Parsia, B. and Labrou, Y. (2003). Task computing - the semantic web meets pervasive computing, *International Semantic Web Conference (ISWC '03)*, Vol. 2870, Springer-Verlag, pp. 866 – 881.

McIlraith, S. A., Son, T. C. and Zeng, H. (2001). Semantic web services, *Intelligent Systems, IEEE* **6**(2): 46 – 53.

Mokhtar, S. B., Preuveneers, D., Georgantas, N. and Issarny, V. (2008). EASY: Efficient SemAntic Service DiscoverY in pervasive computing environments with QoS and context support, *Journal Of System and Software, Elsevier* **81**(5): 785 – 808.

Moore, D. (2006). *Basic Practice of Statistics*, 2nd edn, ISBN: 0716774631, WH Freeman.

Motik, B., Shearer, R. and Horrocks, I. (2009). Hypertableau reasoning for description logics, *Journal of Artificial Intelligence Research* **36**: 165 – 228.

Nedos, A., Singh, K. and Clarke, S. (2006). Mobile ad hoc services: Semantic service discovery in mobile ad hoc networks, *Service-Oriented Computing (ICSOC '06)*, Vol. 4294, Springer-Verlag, Chicago, IL, USA, pp. 90 – 103.

Niazi, R. and Mahmoud, Q. H. (2009). An ontology-based framework for discovering mobile services, *7th Annual Communication Networks and Services Research Conference (CNSR '09)*, IEEE, Moncton, BC, Canada, pp. 178 – 184.

O'Donnel, J., Hall, C. and Page, R. (2006). *Discrete Mathematics Using a Computer*, 2nd edn, ISBN: 1-85233-089-9, Springer-Verlag.

Ott, J. and Kutscher., D. (2004). Why seamless? towards exploiting wlan-based intermittent connectivity on the road, *Trans-European Research and Education Networking Association (TERENA) Networking Conference (TNC '04)*, Rhodes, Greece.
URL: *http://www.terena.org/publications/tnc2004-proceedings/papers/ott.pdf (accessed March 2010)*

Palmer, S. (2001). The semantic web: An introduction.
URL: *http://infomesh.net/2001/swintro (accessed: May 2009)*

Paolucci, M., Kawamura, T., Payne, T. R. and Sycara, K. (2002). Semantic matching of web service capabilities, *1st International Semantic Web Conference (ISWC '02)*, Vol. 2342, Springer-Verlag, Sardinia, Italy, pp. 333 – 347.

Patel, P. and Chaudhary, S. (2009). Context aware semantic service discovery, *World Conference on Services - II (SERVICES-2 '09)*, IEEE, Bangalore, pp. 1 – 8.

Patel-Schneider, P. F., McGuiness, D. L., Brachman, R. J., Resnick, L. A. and Borgida, A. (1991). The classic knowledge representation system: Guiding principles and implementation rationale, *SIGART Bulletin, ACM* **2**: 108 – 113.

Peng, R., Mi, Z. and Wang, L. (2008). An OWL-S based adaptive service discovery algorithm for mobile users, *4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, IEEE, Dalian, China, pp. 1 – 5.

Preuveneers, D. and Berbers, Y. (2008a). Encoding semantic awareness in resource-constrained devices, *Intelligent Systems, IEEE* **23**(2): 26 – 33.

Preuveneers, D. and Berbers, Y. (2008b). Pervasive services on the move: Smart service diffusion on the OSGi framework, *Ubiquitous Intelligence and Computing*, Springer-Verlag, pp. 46 – 60.

Ragone, A., Straccia, U., Bobillo, F., Noia, T. D., Sciascio, E. D. and Donini, F. M. (2008). Fuzzy description logics for bilateral matchmaking in e-marketplaces, *International Workshop on Description Logic (DL 2008)*, CEUR-WS.org, Dresden, Germany.
URL: *http://gaia.isti.cnr.it/ straccia/download/papers/DL08/DL08.pdf (accessed March 2010)*

Raman, R., Livny, M. and Solomon, M. (1999). Matchmaking: An extensible framework for distributed resource management, *Cluster Computing, Springer-Verlag* **2**(2): 129 – 138.

Ranganathan, A. and Campbell, R. H. (2003). A middleware for context-aware agents in ubiquitous computing environments, *ACM/IFIP/USENIX International Middleware Conference (Middleware '03)*, Springer-Verlag, Rio de Janeiro, Brazil, pp. 143 – 161.

Ratsimor, O., Chakraborty, D., Joshi, A. and Finin, T. (2002). Allia: Alliance-based service discovery for ad-hoc environments, *2nd International Workshop on Mobile Commerce*, ACM, Atlanta, Georgia, USA, pp. 1 – 9.

Reaghunathan, V., Schurgers, C., Park, S. and Srivastava, M. B. (2002). Energy-aware wireless microsensor networks, *Signal Processing Magazine, IEEE* **19**(2): 40 – 50.

Rector, A., Bechhofer, S., Goble, C. A., Horrocks, I., Nowlan, W. A. and Solomon, W. D. (1997). The grail concept modelling language for medical terminology, *Artificial Intelligence in Medicine* **9**: 139 – 171.

Resnik, P. (1999). Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language, *Journal of Artificial Intelligence Research* **11**: 95 – 130.

Riva, A. and Ramoni, M. (1996). Lispweb: A specialised HTTP server for distributed AI applications., *Computer Networks and ISDN Systems, Elsevier* **28**(7-11): 953 – 961.

Roto, V. and Oulasvirta, A. (2005). Need for non-visual feedback with long response times in mobile HCI, *14th International World Wide Web Conference Committee (IW3C2)*, ACM, Chiba, Japan, pp. 775 – 781.

Rudolph, S., Krotzsch, M., Hitzler, P., Sintek, M. and Vrandecic, D. (2007). Efficient OWL reasoning with logic programs - evaluations, *Web Reasoning and Rule Systems*, Vol. 4524, Springer-Verlag, pp. 370 – 373.

Rudolph, S., Tserendorj, T. and Hitzler, P. (2008). What is approximate reasoning?, *2nd International Conference on Web Reasoning and Rule Systems*, Vol. 5341, Springer-Verlag, Karlsruhe, Germany, pp. 150 – 164.

Ruta, M., Noia, T. D., Sciascio, E. D. and Scioscia, F. (2008a). Match'n'date: Semantic matchmaking for mobile dating in P2P environments, *2nd International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR '08) in conjunction with the Semantic Web Conference (ISWC '08)*, CEUR-WS.org, Karlsruhe, Germany, pp. 83 – 97.

Ruta, M., Noia, T. D., Sciascio, E. D. and Scioscia, F. (2008b). A semantic-enabled mobile directory service for RFID-based logistics applications, *International Conference on e-Business Engineering (ICEBE '08)*, IEEE, pp. 333 – 340.

Sailhan, F. and Issarny, V. (2005). Scalable service discovery for MANET, *3rd International Conference on Pervasive Computing and Communications (PerCom '05)*, IEEE, Kauai Island, Hawaii, pp. 235 – 244.

Sattler, U. (1996). A concept language extended with different kinds of transitive roles, *20th German Annual Conference on Artificial Intelligence (KI '96)*, Vol. 1137, Springer-Verlag, Dresden, Germany, pp. 333 – 345.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges, *Personal Communications, IEEE* **8**: 10 – 17.

Schlobach, S., Blaauw, E., Kebir, M. E., ten Teije, A., van Harmelen, F., Bortoli, S., Hobbelman, M. C., Millian, K., Ren, Y., Stam, S., Thomassen, P., van het Schip, R. C. and van Willigem, W. (2007). Anytime classification by ontology approximation, *1st Workshop on New forms of Reasoning for the Semantic Web: Scalable, Tolerant and Dynamic in-conjunction with International Semantic Web Conference (ISWC '07) and Asian Semantic Web Conference (ASWC '07)*, Vol. 291, CEUR-WS.org, Busan, Korea.
URL: *http://www.cs.vu.nl/ frankh/postscript/ISWC07-WS.pdf (accessed March 2010)*

Schmidt, H., Kohrer, A. and Hauck, F. J. (2008). SoapME: A lightweight JavaME web service container, *3rd Workshop on Middleware for Service Oriented Computing (MW4SOC '08) in conjunction with the ACM/IFIP/USENIX International Middleware Conference*, ACM, Leuven, Belgium, pp. 13 – 18.

Sheth, A. P., Ramakrishnan, C. and Thomas, C. (2005). Semantics for the semantic web: The implicit, the formal and the powerful, *International Journal on Semantic Web Information Systems, IGI* **1**(1): 1 – 18.

Shim, Y.-S., Kim, Y.-S. and Lee, K.-H. (2009). A mobility-based clustering and discovery of web services in mobile ad-hoc networks, *International Conference on Web Services (ICWS '09)*, IEEE, Los Angeles, CA, pp. 374 – 380.

Singh, M. P. and Huhns, M. N. (2005). *Service-Oriented Computing Semantics, Processes, Agents*, ISBN: 9780470091487, John Wiley and Sons.

Singh, S., Puradkar, S. and Lee, Y. (2005). Ubiquitous computing: Connecting pervasive computing through semantic web, *Information Systems and eBusiness Management Journal, Springer-Verlag* **4**(4): 421 – 439.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. and Katz, Y. (2007). Pellet: A practical OWL-DL reasoner, *Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier* **5**(2): 51 – 53.

Skoutas, D., Simitsis, A. and Sellis, T. (2007). A ranking mechanism for semantic web service discovery, *4th International Workshop on Services Computing Workshops (SCW '07)*, IEEE, Salt Lake City, Utah, USA, pp. 41–48.
URL: *http://www.dbnet.ece.ntua.gr/pubs/uploads/TR-2007-7.pdf (accessed March 2010)*

Smith, D. E., Genesereth, M. R. and Ginsberg, M. L. (1986). Controlling recursive inference, *Artificial Intelligence* **30**: 343 – 389.

Sridhar, V. (2007). Analysis of inter-regional mobile services growth in india, *6th Conference on Telecommunication Techno-Economics (CTTE '07)*, Helsinki, Finland, pp. 1 – 6.

Srinivasan, N., Paolucci, M. and Sycara, K. (2005). Semantic web service discovery in the OWL-S IDE, *39th International Conference on System Sciences*, Vol. 6, IEEE, Hawaii, USA, pp. 109 – 119.

Srirama, S. N., Jarke, M. and Parinz, W. (2006). Mobile web service provisioning, *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW)*, IEEE, pp. 120 – 126.

Srirama, S. N., Jarke, M. and Prinz, W. (2007). Mobile web services mediation framework, *2nd workshop on Middleware for service oriented computing in*

conjunction with the ACM/IFIP/USENIX International Middleware Conference (MW4SOC '07), Newport Beach, California, pp. 6 – 11.

Srirama, S. N., Jarke, M., Zhu, H. and Prinz, W. (2008). Scalable mobile web service discovery in peer to peer networks, *3rd International Conference on Internet and Web Applications and Services (ICIW)*, IEEE, pp. 668 – 674.

Steele, G. L. (1990). *Common Lisp*, 2nd edn, ISBN: 1-55558-041-6, Digital Press.

Steller, A. L., Krishnaswamy, S. and Gaber, M. (2009a). A weighted approach to partial matching for mobile reasoning, *Proceedings of the 8th International Semantic Web Confidence (ISWC '09)*, Vol. 5823, Springer-Verlag, Virginia, USA, pp. 618 – 633.

Steller, L. and Krishnaswamy, S. (2008a). Optimised mobile reasoning for pervasive service discovery, *International Conference on Web Services (ICWS '08)*, IEEE, Beijing, China, pp. 789 – 790.

Steller, L. and Krishnaswamy, S. (2008b). Optimised semantic reasoning for pervasive service discovery, *International Conference on Service Oriented Computing (ICSOC '08)*, Vol. 5364, Springer-Verlag, Sydney, Australia, pp. 620 – 625.

Steller, L. and Krishnaswamy, S. (2008c). Pervasive service discovery: mTableaux mobile reasoning, *International Conference on Semantic Systems (I-SEMANTICS '08)*, Graz, Austria, pp. 93 – 101.

Steller, L. and Krishnaswamy, S. (2009). Efficient mobile reasoning for pervasive discovery, *Symposium on Applied Computing (SAC '09), Session on Semantic Web and Applications*, ACM, Honolulu, Hawaii, USA, pp. 1247 – 1251.

Steller, L., Krishnaswamy, S., Cuce, S., Newmarch, J. and Loke, S. (2008). A weighted approach for optimised reasoning for pervasive service discovery

using semantics and context, *10th International Conference on Enterprise Information Systems (ICEIS '08)*, Barcelona, Spain, pp. 113 – 118.

Steller, L., Krishnaswamy, S. and Gaber, M. (2009b). Cost efficient, adaptive reasoning strategies for pervasive service discovery, *International Conference on Pervasive Services (ICPS '09)*, ACM, London, UK, pp. 11 – 20.

Steller, L., Krishnaswamy, S. and Gaber, M. M. (2009c). Enabling scalable semantic reasoning for mobile services, *International Journal of Semantic Web Information Systems - Special Issue on Scalability and Performance of Semantic Systems, Vassilis Christophides, Stefan Decker and Jeff Heflin (Eds.), IGI* **5**(2): 91 – 116.

Steller, L., Krishnaswamy, S. and Newmarch, J. (2006). Discovering relevant services in pervasive environments using semantics and context, *3rd International Workshop on Ubiquitous Computing (IWUC-2006). In conjunction with ICEIS 2006.*, INSTICC Press, Paphos, Cyprus, pp. 3 – 12.

Stoilos, G., Stamou, G. B. and Pan, J. Z. (2008). Classifying fuzzy subsumption in fuzzy-EL+, *International Workshop on Description Logics (DL2008)*, CEUR-WS.org, Dresden, Germany.
URL: *http://ceur-ws.org/Vol-353/StoilosStamouPan.pdf (accessed March 2010)*

Straccia, U. (2005). Towards a fuzzy description logic for the semantic web (preliminary report), *2nd European Semantic Web Conference (ESWC '05)*, Vol. 3532, Springer-Verlag, Heraklion, Crete, Greece, pp. 167 – 181.

Stuckenschmidt, H. and Harmelen, F. v. (2002). Approximating terminological queries, *International Conference on Flexible Query Answering Systems (FQAS '02)*, Vol. 2522, Springer-Verlag, Copenhagen, Denmark, pp. 329 – 343.

Stuckenschmidt, H. and Klein, M. (2004). Structure-based partitioning of large concept hierarchies, *International Semantic Web Conference (ISWC '04)*, Vol. 3298, Springer-Verlag, Hiroshima, Japan, pp. 289 – 303.

Stuckenschmidt, H. and Kolb, M. (2008). Partial matchmaking for complex product and service descriptions, *Multikonferenz Wirtschaftsinformatik (MKWI '08)*, GITO-Verlag, Munich, Germany.
URL: *http://webrum.uni-mannheim.de/math/lski/public/MKWI08-SWBIS.pdf (accessed March 2010)*

Stuckenschmidt, H. and Schlicht, A. (2009). Structure-based partitioning of large ontologies, *Modular Ontologies, Springer-Verlag* **5445**: 187 – 210.

Su, J. and Guo, W. (2008). A survey of service discovery protocols for mobile ad-hoc networks, *International Conference on Communications, Circuits and Systems (ICCCAS '08)*, IEEE, Fujian, China, pp. 398 – 404.

Suraci, V., Mignanti, S. and Aiuto, A. (2007). Context-aware semantic service discovery, *16th IST Mobile and Wireless Communications Summit*, IEEE, pp. 1 – 5.

Sycara, K., Widoff, S., Klusch, M. and Lu, J. (2002). LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace, *Autonomous Agents and Multi-Agent Systems, Kluwer Academic* **5**(2): 173 – 203.

SysCon (2005). Microsoft, IBM, SAP to discontinue UDDI web services registry effort, SOA World Mazagine.
URL: *http://soa.sys-con.com/node/164624 (accessed May 2009)*

Tergujeff, R., Haajanen, J., Leppanen, J. and Toivonen, S. (2007). Mobile SOA: Service orientation on lightweight mobile devices, *International Conference on Web Services (ICWS '07)*, IEEE, Salt Lake City, USA, pp. 1224 – 1225.

Thanh, D. V. and Jorstad, I. (2005). A service-oriented architecture framework for mobile services, *Advanced Industrial Conference*

on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/ E-Learning on Telecommunications Workshop (AICT/SAPIR/ELETE), IEEE, pp. 65 – 70.

Toninelli, A., Corradia, A. and Montanaria, R. (2008). Semantic-based discovery to support mobile context-aware service access, *Computer Communications, Mobility Management and Wireless Access, Elsevier* **31**(5): 935 – 949.

Trastour, D., Bartolini, C. and Gonzalez-Castillo, J. (2001). A semantic web approach to service description for matchmaking of services, *1st International Semantic Web Working Symposium (SWWS '01)*, Stanford University, California, USA, pp. 447 – 461.

Tsarkov, D. and Horrocks, I. (2005). Ordering heuristics for description logic reasoning, *19th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 609 – 614.

Tsarkov, D. and Horrocks, I. (2006). FaCT++ description logic reasoner: System description, *3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, Vol. 4130, Springer-Verlag, Seattle, WA, USA, pp. 292 – 297.

Tsarkov, D., Horrocks, I. and Patel-Schneider, P. F. (2007). Optimizing terminological reasoning for expressive description logics, *Journal of Automated Reasoning, Springer-Verlag* **39**(3): 277 – 316.

Veijalainen, J. (2008). Mobile ontologies: Concept, development, usage and business potential, *International Journal on Semantic Web and Information Systems, IGI* **4**(1): 20 – 34.

Veijalainen, J., Nikitin, S. and Tormala, V. (2006). Ontology-based semantic web service platform in mobile environments, *7th International Conference on Mobile Data Management (MDM '06)*, IEEE, pp. 83 – 91.

Ververidis, C. and Polyzos, G. (2005). Extended ZRP: A routing layer based service discovery protocol for mobile ad hoc networks, *2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous '05)*, IEEE, San Diego, California, pp. 65 – 72.

Vu, L.-H., Porto, F., Aberer, K. and Hauswirth, M. (2007). An extensible and personalized approach to QoS-enabled service discovery, *11th International Database Engineering and Applications Symposium (IDEAS '07)*, IEEE, Banff, Alberta, Canada, pp. 37 – 45.

Vu, L., Hauswirth, M. and Aberer, K. (2005). QoS-based service selection and ranking with trust and reputation management, *13th International Conference on Cooperative Information Systems (CoopIS '05)*, Agia Napa, Cyprus, pp. 466 – 483.

W3C (2004). OWL web ontology language.
URL: *http://www.w3.org/TR/2004/REC-owl-features-20040210/ (accessed December 2009)*

Wache, H., Groot, P. and Stuckenschmidt, H. (2005). Scalable instance retrieval for the semantic web by approximation, *International Workshops on Web Information Systems Engineering (WISE '05)*, Vol. 3807, Springer-Verlag, New York, NY, USA, pp. 245 – 254.

Wache, H., Serafini, L. and Garcia-Castro, R. (2004). Survey of scalability techniques for reasoning with ontologies, *Deliverable of EU-Project KnowledgeWeb*, pp. 1 – 81.
URL: *http://www.google.com.au/interstitial?url=http://www.jarrar.info/-publications/D2.1.1-Scalability.pdf (accessed February 2010)*

Wang, Z. and Hu, Y. (2008). An approach for semantic web service discovery based on p2p network, *4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, IEEE, Dalian,

China, pp. 1 – 4.

Wei, Z., Kang, M. and Zhou, W. (2008). A semantic web-based enterprise information integration platform for mobile commerce, *International Conference on Management of e-Commerce and e-Government (ICMECG '08)*, IEEE, Jiangxi, China, pp. 57 – 60.

Weiser, M. (1991). The computer of the 21st century, *Scientific American* **3**(265): 66 – 75.

Wolowski, V., Ishikawa, N. and Sumino, H. (2007). Semantic web approach to content personalization, *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM '07)*, IEEE, Washington, DC, USA, pp. 109 – 117.

Xiaosu, C. and Jian, L. (2005). Build mobile services on service oriented structure, *International Conference on Wireless Communications, Networking and Mobile Computing*, Vol. 2, IEEE, pp. 1472 – 1476.

Zabariadis, T. and Doshi, B. (2004). Applications and services for the B3G/4G era, *Wireless Communications, IEEE* **11**(5): 3 – 5.

Zacharias, V., Abecker, A., Vrandecic, D., Borgi, I., Braun, S. and Schmidt, A. (2007). Mind the web, *1st Workshop on New forms of Reasoning for the Semantic Web: Scalable, Tolerant and Dynamic in-conjunction with International Semantic Web Conference (ISWC '07) and Asian Semantic Web Conference (ASWC '07)*, Vol. 291, CEUR-WS.org, Busan, Korea.
URL: *http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-291/paper08.pdf (accessed March 2010)*

Zadeh, L. (1965). Fuzzy sets, *Information and Control* **8**(3): 338 – 353.

Zadeh, L. A. (1975). Fuzzy logic and approximate reasoning, *Synthese, World Scientific Publishing* **30**(3-4): 407 – 428.

Zoric, J., Gjermundshaug, N. and Alapnes, S. (2007a). Experiments with semantic support in mobile service architectures, *65th Vehicular Technology Conference (VTC '07)*, IEEE, Dublin, Ireland, pp. 292 – 297.

Zoric, J., Gjermundshaug, N. and Alapnes, S. (2007b). Service mobility a challenge for semantic support, *16th IST Mobile and Wireless Communications Summit*, IEEE, Budapest, Hungary, pp. 1 – 7.