An infrastructure mechanism for dynamic ontology-based knowledge infrastructures

Maciej Zurawski

Doctor of Philosophy

Centre for Intelligent Systems and Their Applications

School of Informatics

University of Edinburgh

2009

Table of Contents

Thesis Abstract

Both semantic web applications and individuals are in need of knowledge infrastructures that can be used in dynamic and distributed environments where autonomous entities create knowledge and build their own view of a domain. The prevailing view today is that the process of ontology evolution is difficult to monitor and control, so few efforts have been made to support such a controlled process formally involving several ontologies. The new paradigm we propose is to use an infrastructure mechanism that processes ontology change proposals from autonomous entities while maintaining user-defined consistency between the ontologies of these entities. A core invention of our approach is to formalise consistency constraints as so called *spheres of consistency* that define 1) knowledge regions within which consistency is maintained and 2) a variable degree of proof-bounded consistency within these regions. Our infrastructure formalism defines a protocol and its computational semantics, as well as a model theory and proof theory for the reasoning layer of the mechanism. The conclusion of this thesis is that this new paradigm is possible and beneficial, assuming that the knowledge representation is kept simple, the ontology evolution operations are kept simple and one proposal is processed at a time.

Acknowledgements

I am profoundly grateful to my supervisors Prof. Dave Robertson and Dr. Alan Smaill for their support, guidance and assistance during the long course of this thesis.

I am also grateful to my family for taking care of me during the period of 1.5 years when I was very ill and had to take a long break from my research. With their help, I managed to recover so that I could finish this piece of work.

Finally, I would like to thank my funding agencies from Sweden: The Marcus Wallenberg Foundation for Education in International Industrial Enterprise, The Foundation Blanceflor Boncompagni-Ludovisi née Bildt, The Hans-Werthén foundation (governed by the Royal Swedish Academy of Engineering Sciences, IVA), Helge Ax:son Johnsons Foundation, and The Lars Hierta Memorial Foundation.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

....

(Maciej Zurawski)

Chapter 1 Introduction and Motivation

1.1 Can Technology Help Organizing and Communicating Our Knowledge?

Communication holds society together. Often, the purpose is to communicate a form of knowledge in order to facilitate some activity. In the early phases of our human history, the content of that knowledge was kept in our memory and communicated using ordinary language, when it was needed. Later, when writing was invented, this communication could happen in written form, e.g. by sending a letter. In both oral and written communication, it is the responsibility of the speaker or writer to investigate what language the recipient understand and express the message in a way adapted to the recipient. Because there are many languages and sub-cultures, this process requires a substantial effort from the sender.

When efficiency and productivity become important in society, a tendency developed to streamline this communication language in order to facilitate communication efficiency. However, when there is a lot of change in society, then this language must constantly adapt to changing circumstances. These are two conflicting requirements that have to be balanced. In the information society, computer systems and networks could be used to communicate written words. However, as before, even then this requires extensive amounts of effort both from the human sender and the human recipient. Ever increasing demands on efficiency and productivity has led to the knowledge society, where knowledge technologies are used to codifying knowledge in a way that makes it *computer understandable*. Then the processing of communicating structured knowledge becomes more efficient.

This approach requires, however, that every communicating group or entity formalises a formal model of how it sees the world[1], because it will use that model for sending or interpreting formally codified knowledge. Because the world is a complex and constantly changing place, these models vary from group to group, and they also change. This poses a big challenge to the benefits of using knowledge technology. Two things are therefore

[1] The alternative is that all groups use the same model of the world, and if that model is hard-wired in their communication protocol it does not have to be represented explicitly, but it will then be very difficult to modify the world model.

needed. The first one is a macro theory of how semantics (both general concepts and instances of these) are used by various entities that constantly evolve their world models, at the same time as *some* pairs or groups of these entities still need to understand each other, perfectly or mostly. The second thing needed is a technical system that maintains this dynamic process where knowledge language of entities evolves, but still maintains knowledge language coherence between the selected pairs or groups of entities that need to exchange knowledge for the purpose of co-ordinating their activities.
This thesis aims at contributing to these two points.

1.2 Basic Notions

We have explained that human knowledge can become codified and we have introduced the notion of formal models about the world, used by a certain entity or group. These are called *ontologies*, and (Noy et al., 2001) defines an ontology as being a formal explicit description of concepts in a domain of discourse. This formal description will be assumed to use some form of logic. The smallest elements of an ontology are concepts and from a formal point of view they have a specific meaning (see chapter 3), but we will say something more about their social and pragmatic meaning. Often, knowledge is implicitly stored in documents written using ordinary language – but that granularity or lack of formality is not good enough for automated processing of knowledge (Staab et al. 2001). Therefore, in this paradigm we advocate that a system should instead store individual pieces of knowledge (using concepts from ontologies) because it can then process and compare these pieces of knowledge.

1.3 Knowledge Management as a Motivation

Knowledge Management (Kingston, 2000) is the study of how knowledge can be reused in organizations and how it can be provided in the right moment to the right person that e.g. is making a decision where this knowledge is needed. The *knowledge organization* (Bennet 2003) is a vision of the organization of the future (as projected for 2020 A.D.) that is viewed as an intelligent complex adaptive system. It is envisioned as being dynamic and flexible, but it also needs some *integrative forces* that keep the knowledge system and the organization together. Knowledge Management is the original motivation for our research but we later saw that the outcome potentially can be applied to other areas as well.

1.3.1 The Benefits of Ontology-Based Knowledge Management

Unfortunately, it has been discovered (Everett 2002) that if a system that consists of information or knowledge in the form of unprocessed text and the collections grows, redundancy can make the system *unusable*. They describe a system that stores documents in text format, but by formalizing them and using ontologies, then matching between the documents can be used and the redundancy can be eliminated. Ontologies are therefore useful for knowledge management and ontologies and less formalised documents can be combined in the same system (Reimer 1998).

1.3.2 Distributed Knowledge Management

Two of the reasons why the act of forcing an organization to use centralized ontologies fails (Bonifacio et al., 2002) are that users might see them as irrelevant (and not possible to understand deeply) or oppressive (i.e. the users disagree with the schemas). Distributed Knowledge Management (DKM) investigates how knowledge should be managed in a decentralised organization that has several divisions and that is not governed by central control.

Ontologies could be useful in knowledge management, and in the case of DKM we argue that ontologies should be adapted to the local needs and business requirements of every division, because then the *autonomy* of that division is respected. Also, this helps to minimise the *cognitive distance,* i.e. the difference between an ontology used and the internal model somebody has of reality. It is then clear that for the purpose of this application, several ontologies should exist and be linked by *ontology mappings* instead of being merged into a single ontology.

It has been argued (Bonifacio et al., 2002) that technological and social architectures must be consistent. Also, the organizational structure should be consistent with the technology used. In most KM software solutions a single big schema of concepts and objects is enforced on the people involved, whereas all contextual and subjective aspects or knowledge creation are eliminated. Therefore, the subjective and social aspects of knowledge creation and modelling are lost. There are two kinds of KM architectures:

Centralised KM architectures - these focus on refining and validating knowledge in a way that will create a single official knowledge structure. This can involve using an Enterprise Knowledge Portal. In a study ((Bonifacio et al., 2000) such systems were deserted by end-users. The two reasons for failure were: the schemas were perceived as irrelevant or as oppressive. However, in limited contexts, e.g. when there is a consensus about the business knowledge or there is an authority governing and controlling it, centralised KM architectures are used.

Distributed KM architectures – these technologies support the autonomous creation and organization of knowledge locally produced by individuals and groups, but it also supports coordination processes. It *decentralizes control* to individuals or local groups. It supports the evolution of emergent semantics that appear bottom-up rather than top-down.

There are situations where these two approaches can be combined (e.g. Desouza et al. 2004 describes such a combined system).

We will now mention two basic principles.

1.3.3 Two fundamental principles

These two philosophical principles were described informally by (Bonifacio et al., 2002) and they have influenced our computational formalism and technical solution:

Principle of Autonomy – each community has a high degree of autonomy to manage its local knowledge. They should particularly have semantic autonomy, defined as "the possibility of choosing the most appropriate conceptualizations of what is locally known".

Principle of Coordination – communities adopts mechanisms for meaning translation across different interpretative contexts, called semantic inter-operation.

The question is if the notion of semantic autonomy can become more computational? Section 1.5 will explore this.

1.4 Knowledge creation is a collaborative process

We agree with other researchers (Froehner et al. 2004, Euzenat 1996) that modelling the social and collaborative *process* of how meaning is created is important. The social process supporting knowledge creation and communication among many autonomous actors has been rather neglected in the ontology community. E.g. in business-to-business integration (i.e. the formalised exchange of messages between enterprises using a certain exchange format, see Bussler 2001) one needs to use systems that translate from the business transactions (e.g. orders) in one company to a system in another company and ontology mappings could be used for this. However, is it possible to fix the problem by creating a "standard" ontology that businesses are supposed to use? This solution is not viable as an ultimate one, because the business world is constantly changing and therefore even here one needs a process that has the freedom to evolve the semantics of the ontology, but in way that does not break down the communication with important parties.

However, almost all communication formalisms assume that the language used for communicating is fixed. But there must also be a process that changes the language (the ontology) and coordinates these changes with the involved entities. This leads us to the generalised definition of semantic autonomy.

1.5 Semantic Autonomy

In our work (Zurawski, 2004) we developed requirements on a distributed KM system for a decentralised organization. These requirements are related to the discussion above and can be grouped in the categories of 1) freedom of the organizations' divisions to evolve their ontologies, and 2) maintaining coherence of the organization's knowledge so that it becomes

Table 1. The generalised definition of semantic autonomy

The definition of *Semantic autonomy* requires these properties to hold:

1. The local contexts have the freedom to propose a change in their local ontology (i.e. the ontology of the local context) or in the mappings from their ontology. All the possible request types, operation types and explicit change process that manage them, are explicated and formalised.

2. The system does “in some way” maintain full or bounded consistency as defined within every sphere of consistency.

3. The ontological language is dynamic and open-ended (i.e. not confined by a pre-defined set) but there is a knowledge source that can provide knowledge in this language and about its evolution.

"well-behaved" when an outsider interacts with it. That motivation inspired this generalised definition of semantic autonomy (Zurawski et al., 2008):

"Contexts" can be formalised in many different ways but they are normally seen as passive logical constructions. In the definition above, "contexts" are therefore fundamentally different from the usual definition because in addition to that logical connotation they have inherited some "agent-like" behaviour: the can initiate action at the knowledge level. Therefore, we could have called them agents, but because that implies many other abilities as well, have chosen the descriptor "contexts".
The definition uses the term "spheres of consistency" and we can at this stage understand this as regions of agreement about how several ontologies should be translated between each other. This can be a *conflict-free* area of agreement (or it can also allow for "some" minor conflicts as we will see). These notions will later be fully formalised. The second point of this definition will be called "semantic reliability", because it is possible to define constraints that are maintained in order to facilitate interoperability and eliminate/reduce miscommunication in the system (more precisely, this can be seen as unsoundness that occurs in query-answering after ontology mappings have been used for translating knowledge).

1.6 Where is there reliability in an open world?

We have defined semantic reliability for an important reason. If there is a misunderstanding in the communication, i.e. an error in translation from one ontology to another one, then the users or applications relying on that translation might crash or in the optimistic case, try to circumvent this problem by re-trying the translation process. Semantic applications are applications that rely on and utilise ontologies. Translation of knowledge from one ontology to another one is done via mappings. In some domains, this process must be fully reliable, otherwise the users are forced to use some more old-fashioned methods for ensuring semantic reliability.

An "open world" e.g. like the Web or Semantic Web (Heflin and Hendler, 2000) is characterised by many ontologies controlled by various autonomous entities and the ontological language that these ontologies can use is open-ended. One could try to create reliability by mapping all local ontologies to a shared one (Stuckenschmidt et al. 2002) but that shared ontology becomes then difficult to maintain. One could also try to create reliability of some form by forcing the users to only use one type of software for managing ontologies (see e.g. Maedche, et al., 2003) but that goes against the spirit of an open environment because an open environment should support various forms of software or services. Also, it is difficult to receive reliability from a piece of software if it does not have a formal specification of what it does. Today, (May 2009) there is no ontology management system that formally specifies its *processes*, computational semantics and semantics of its logic.

We therefore need a formally specified technical system, that has semantic autonomy, about which we can prove certain properties. That will give us the best reliability while still being facilitating an "open world" where various forms of software and services can implement the same specification, and where various ontologies are evolving.

1.7 A semantic macro theory

There are semantic theories that focus on the small scale, e.g. how to represent knowledge about an event using an ontology and how to reason within that ontology or how to evolve that ontology. In chapter 2 we mention some theories that investigate reasoning with several ontologies. There is also research about data integration and data federation.

However, do we need any particular theories for modelling the dynamics at the large scale of ontology-based systems where change is initiated by many autonomous entities?
For example (Parsia et al. 2006) has argued that we should consider how the meaning on the Semanitc Web as a whole is created. They mention two different theories of meaning. The first one is that meaning is what is intended by the user, and second one is that meaning is what is defined by ontological documents. In the second case, this could vary between only being dependent on a local document (e.g. containing an ontology), and between being dependent on all documents on the whole web. An intermediary solution is that the meaning is defined by a document and the relevant connected documents.

When we consider this macroscopic semantic level, we feel that we yet have not come across a *computational* theory that models ontologies and their change, that models how change is initiated, that models which ontologies are connected with each other and how groups of ontologies maintain coherence. We have therefore in this thesis developed an initial version of such theory (see chapters 3, 4 and 5) and chapter 9 contains a vision for its future

development, whereas chapter 2 illustrates which related theories we have come across. Some of them have inspired us, whereas others represent opposing views of how to approach this problem, but we have presented them as well for the sake of comparison.
Because this theory is computational, it can be implemented in software and chapter 6 describes our implemented prototype. Because this technology is not an end in itself, but a *facilitator* for other applications that utilize ontologies, and also a host of ontologies, we have called it an infrastructure mechanism.

1.8 Why do we need knowledge infrastructures?

The infrastructure mechanism is the heart of a knowledge infrastructure.
A *knowledge infrastructure*:

- Is a system for the automated management and evolution of several ontologies and mappings between them.
- Serves as the basis for semantic applications, and should provide some underlying reliability (as discussed in §1.6).
- Has its core specified by an infrastructure mechanism.
- Could contain other elements, such as a graphical interface (see e.g. §2.2.3), but that is not the focus of this thesis.

Why are knowledge infrastructures needed?

- Because semantic applications often assume the existence of an ontology or several ontologies and mappings that are already set up.
- Because semantic applications should focus on their core activities rather than managing networked ontologies. However, they could send messages to a knowledge infrastructure containing a *proposal* to change an ontology or ontology mapping.
- Ontologies are often not owned by a single application but used by several semantic applications.

In our published work (Zurawski 2006), we developed a framework and prototype of an infrastructure mechanism that maintains the autonomy of several entities proposing changes to their ontologies, while still maintaining full consistency between them by means of following certain rule-based policies. However, in our later work (Zurawski, Smaill and Robertson 2008), we provided a summary of a generalised infrastructure mechanism (no longer only supporting full consistency of the whole system) and how it works in a simple scenario where the degree of consistency is changing. This thesis illustrates the full details of this infrastructure mechanism.

1.9 How are semantic applications and knowledge infrastructures connected?

Let us envision the following semantic application. An e-commerce shop sells various kinds

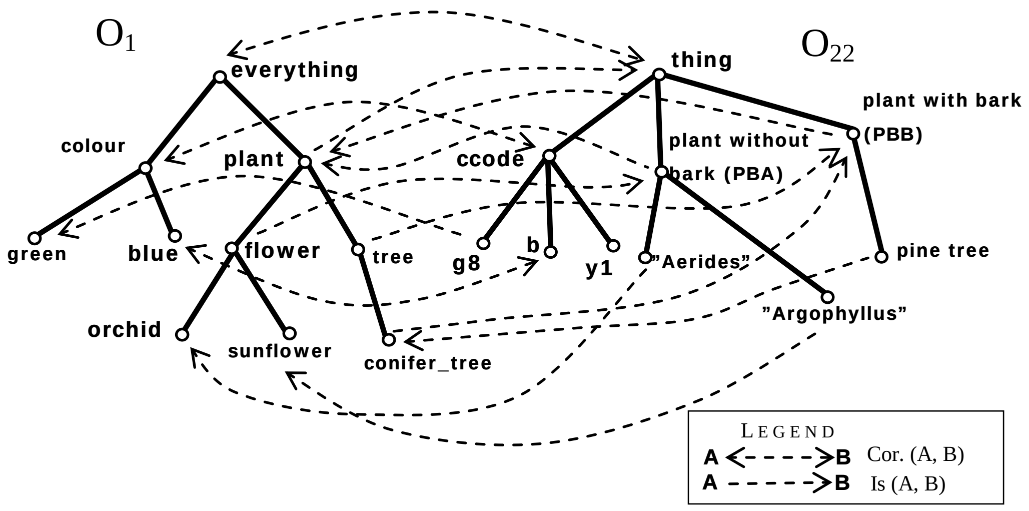


Figure 1. The ontologies of the two divisions of a shop.

of plants and it has two divisions, one is the customer facing one and one division is concerned with "product development", in this case buying and classifying new plants. The two divisions have their own ontologies (see Figure 1) because they need to have the flexibility to adapt to their own needs. The e-commerce shop could be negotiating with a customer (i.e. this negotiation is the core of this application) and the customer might ask "Do you have yellow sunflowers?". This enquiry is translated to formal terms (or it could have been done using formal terms to start with) and customer department will then discover that it does not have "yellow" as a concept in its ontology. After some work it finds out that it should become a type of "colour" in its ontology. Because that department needs to able to talk with customers, it sends a *proposal request* to the infrastructure mechanism that it wants to add the concept of "yellow" and that it should be a type of "colour". The infrastructure mechanism then investigates if this request can be accommodated or if it would create some form of disorder in the knowledge of the whole organization – in this case two connected ontologies. The infrastructure mechanisms sends back a verification that this change would not create any such problems. Some more internal processing leads to the insight that the concept "y1" in the product department actually means the same thing as "yellow" in the customer department. In other words, the product development had already considered this as a potential product feature, but it was not communicated to the customer department before. Therefore, a new request is sent to knowledge infrastructure by the semantic application, and this new request proposes that "yellow" should correspond to "y1" – i.e. an ontology mapping. The infrastructure mechanism processes that proposals, investigates the logical consequences of such change (which are unproblematic in this case) and asks both divisions if they would accept such change. This leads to the change being done.

Now the semantic application can return to its "core activity" and continue the negotiation because it can actually understand the question asked by the customer. Perhaps it could answer "Yes, we have yellow sunflowers". It has now also adapted its knowledge representation to its own needs, i.e. the customer needs, and then this was followed by connecting this knowledge with the ontology of its other division.
So we have now illustrated how semantic applications communicate with a knowledge infrastructure, and how they benefit from having access to a knowledge infrastructure.

1.10 The contribution of this thesis.

The contribution of this thesis is a theoretical formalisation of an infrastructure mechanism that explicitly models

1) the distributed process of initiating change
2) how a context proposes to evolve its ontology
3) how mappings are proposed to evolve
4) a layered system that processes these proposal transactions by following explicit policies, and does automated reasoning (that calculates if a proposed change would introduce inconsistency or redundancy)
5) two kinds of consistency constraints that the mechanism guarantees to satisfy.

The consistency constraints are formalised using so called *spheres of consistency* that define

1) knowledge regions within which consistency is maintained and
2) a variable degree of proof-bounded inconsistency within these regions.

Our infrastructure formalism defines a protocol and its computational semantics, as well as a model theory and proof theory for the reasoning layer of the mechanism.

In our work we have proved the formal properties of soundness and partial completeness of the reasoning and termination of the processing for our infrastructure mechanism. The second contribution of the thesis is that we have evaluated the infrastructure mechanism experimentally by doing simulations using an implemented prototype.

The new paradigm we propose is to use an infrastructure mechanism that processes ontology change proposals from autonomous entities while maintaining user-defined consistency between the ontologies of these entities. This enables semantic autonomy and decentralised control while at the same time maintaining interoperability between the ontologies of entities.

Chapter 2 Background and related work

2.1 Background

2.1.1 Ontology evolution

The research by (Noy et al. 2004) gives one possible definition of ontology evolution and according to them it is "The ability to manage ontology changes and their effects by creating and maintaining different variants of the ontology" whereas other researchers (including the author) would agree that it is enough to maintain one variant of the ontology. Noy et al. mention three reasons why ontologies change:

- the domain changes
- the conceptualization changes
- the specification (i.e. the logical language used) changes

The first is common because the reality modeled can change often. The second one means that the perspective from which the modelling is done changes. The third means that the logical formalism changes, so the ontology has to be ported from one logical formalism to another one. Our system supports the first two cases. They mention that there are some differences between ontologies and schemas, e.g. ontologies include semantics that define the meaning of instance data, ontologies are more often reused and more decentralised than schemas.

There are composite operations, e.g. moving a concept is equivalent to deleting and re-creating it. They mention *traced evolution,* is when we know what change is made to an ontology and we can consider its consequences on instances and other ontologies. *Untraced evolution* is when we only see two versions of an ontology and we have to infer what changes that were made by identifying similarities and differences between the ontologies. Our system does traced evolution. They also present a long list of operations on classes and slots, e.g. "Create class C".

It is possible to view ontology evolution from a business and user perspective. According to (Stojanovic et al. 2002) "Ontology evolution is the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology based application, as well as the consistent management/propagation of these changes to dependent elements", and we agree with this definition. From this perspective ontology changes are considered both as an organizational and technical process. Instead of direct manipulation of the ontologies, the intent of a change should be expressed. They advocate reversibility and that before one applies a change to the ontology, a list of all consequences for the ontology should be generated and presented. According to them (Stojanovic et al. 2002) there are four stages in the ontology evolution:

1. Representing change
2. Investigating semantics of change
3. Implementation (making the actual change)
4. Propagation (e.g. from one ontology to other)

An interesting distinction that is made is that between top-down vs. bottom-up changes. Bottom-up changes are generated because of instances, i.e. the instances contradict ontological assumptions, and this could mean that the ontology is out of date. Bottom-up change can be proposed automatically by the system, e.g. their KAON API. The Karlsruhe Ontology and Semantic Web framework (KAON) is a platform they use for ontology evolution.

One can differentiate between *structural consistency* which means that a certain logical language is used correctly, *logical consistency* and *user-defined consistency* (Haase et al. 2005). An ontology is logically consistent if it is satisfiable and therefore has a model. An example of user-defined consistency is to disallow redundancy, and that can be done in our framework. According to an alternative approach to ontology evolution (Haase et al. 2005), the user changes the ontology but if it is then inconsistent (in any of the three ways mentioned here) the ontology evolution system does an additional step and uses a resolution function to make an additional change operation such that it can change any inconsistent state into a consistent one. Our approach is different, because we process the intention to change an ontology instead of trying to repair inconsistent ontologies.

According to their approach (Haase et al. 2005), when a conflict occurs in an ontology they present an algorithm to find the “maximal consistent subontology” and “minimal inconsistent subontology". An ontology is a minimal inconsistent subontology if it is inconsistent and if removing any of its axioms would make it consistent. In our formalism we have a related notion because when a proposed change would introduce a contradiction then one possibility is to find a minimal ontology subset that creates the problem and delete it (or propose to delete a single relation).

It is possible to distinguish between changes in the syntax (i.e. some axioms change) or only semantic of ontologies (i.e. only the intrinsic meaning changes) as is done by (Haase et al. 2005b). According to them there are four different ways to handle inconsistency in changing ontologies:

1) Consistent Ontology Evolution
2) Repairing inconsistencies,
3) Reasoning with inconsistent ontologies (by finding consistent subsets)
4) Multi-version reasoning (where all the former versions of an ontology are saved).

Our framework supports 1) and facilitates 3) by maintaining a specified acceptable level of inconsistency. We summarize this section by noting that none of the above-mentioned research actually presents formal process models that describe how the ontologies are allowed to change during different circumstances. However, some of them instead present software for ontology evolution and show their graphical user interface.

2.1.2 Ontology repair and its paradigm

Ontology evolution and ontology repair are related, but ontology evolution does not necessarily need to involve ontology repair. We can consider these as two different paradigms (see Figure 2). In the ontology repair paradigm the ontology moves to a "faulty" state – this is typically due to a new instance that has been created that contradicts other facts or due to a faulty interaction, one where an agent could reach a goal due to some mismatch or miscommunication in the process. In this paradigm, changes to ontologies and instances just happen, there is no system that manages them as such. Instead there is a system for repairing the mismatches and problems (e.g. contradictions) that have occurred as a result of change. This is related to the discussion in §2.1.13 about "living in an unmanaged world" because ontology repair fits well with that paradigm and is useful there.

One example of such a system is presented is Ontology Refinement Systems (ORS) present develop by (McNeill et al. 2005). In their system procedural ontologies are used, that e.g.

describes how an agent can buy a flight ticket (and the pre- and post-conditions of this process). The fundamental assumption is that these ontologies appear in an unmanaged infrastructure, and therefore mismatches can exist between ontologies. Two ontologies could differ because they contain different axioms or predicates. Also, mismatches could occur because of different predicate arities, or different domains or ranges for predicates that otherwise have the same name.

The authors use KIF (Knowledge Interchange Format) as the KR language for representing rich ontologies, and include rules for representing procedures. In this scenario there are agents that have to accomplish certain things, e.g. buy a flight ticket. E.g. there is a rule in the KIF ontology that describes the preconditions and effects that the act of buying a ticket has. However, when the agent is asked to perform an action, sometimes the invocation does not really match the agent's ontology – so the agent's ontology must be refined. This is achieved by translating the KIF ontology to a more basic language, namely PDDL (Planning Domain Definition Language), and then by doing failure analysis of the plan and ontology mismatch, the systems tries to fix the ontology, in a way that removes the problem.

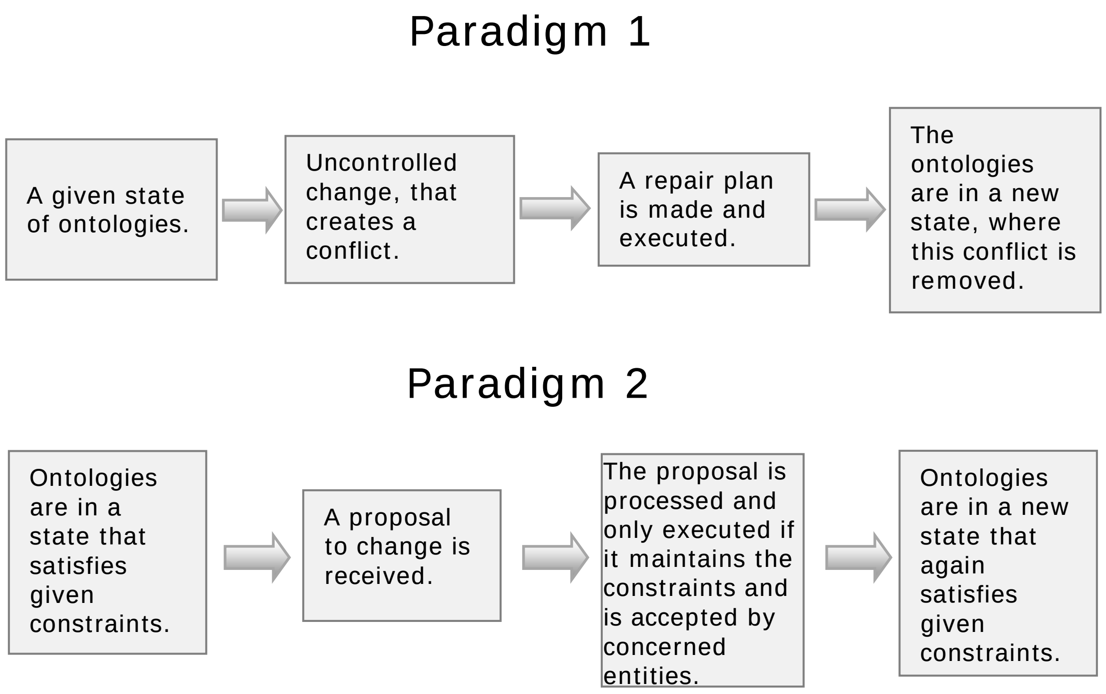

Paradigm 1
A given state of ontologies.
Uncontrolled change, that creates a conflict.
A repair plan is made and executed.
The ontologies are in a new state, where this conflict is removed.
Paradigm 2
Ontologies are in a state that satisfies given constraints.
A proposal to change is received.
The proposal is processed and only executed if it maintains the constraints and is accepted by concerned entities.
Ontologies are in a new state that again satisfies given constraints.

Figure 2. Two ontology evolution paradigms.

This is a contrast to our approach that adheres to very different paradigm, namely that the system does always receive change proposals as such and investigates how these change proposals would affect the consistency of one or several ontologies. Therefore, in our approach the need for repairs is smaller, but in some situations conflicts can still occur, e.g. when new knowledge is correct but conflicts with existing knowledge. In this situation, some of the existing axioms might have to be removed.

2.1.2.1 A technical difference between logical inconsistency and incoherence

At this stage of the discussion we will mention a formal and technical difference between logical inconsistency and logical incoherence as it is defined by (Bell, et al., 2007). To understand this section one should first read Chapter 3, because it discusses subtle details of the logical formalism.
They make these three definitions.

Definition 1 (Unsatisfiable Concept). *A concept name C in an ontology O, is unsatisfiable iff, for each interpretation I of O,* $C^I = \emptyset$.

Definition 2 (Incoherent Ontology). *An ontology O is incoherent iff there exists an unsatisfiable concept name in O.*

Definition 3 (Inconsistent Ontology). An *ontology O is inconsistent iff it has no model.*

These descriptions are done from description-logic perspective, particularly definitions 1 and 2, whereas definition 3 corresponds more to classical logic. Our definition of inconsistency does correspond to definition 3 rather than definition 2. I.e. our infrastructure mechanism would accept certain situations that are logically consistent, but that wouldn't be viewed as coherent according to definition 2, because these situations might require one or more concepts to have no instances (i.e. be unsatisfiable concepts according to the definition above). Chapter 3 will clarify this.

2.1.3 Sources of proposal generation: ontology induction or human insight

In our definition of semantic autonomy in Chapter 1 it is stated that "there is a knowledge source that can provide knowledge in this language and about its evolution." and this means more precisely that something will generate a stream of proposals of how to change and grow ontologies and the mappings between, and by creating these proposals that source will express its knowledge of the domain that one or several ontologies are modelling. The obvious question is where this knowledge will come from. In our approach, we assume that it will either come from human users that share their domain knowledge by expressing it in the form of proposals, or by an automated process such as ontology induction. Ontology induction is a process that takes individual pieces of data or natural language text and from this hypothesizes what relations that exist and how they are related. However, because the process is inductive and often relies on natural language as a source, the knowledge elements it creates cannot typically be proved to be true, but can often be said to likely be true. The knowledge elements that such a process is generating one by one could be used as proposals that are sent individually to the infrastructure mechanism that we define in this thesis. Particularly, many proposals could be used for building the ontologies. Alternatively, humans could make these proposals, or it is also possible to combine these approaches.

2.1.4 The Nature and Origin of Ontology Mappings

2.1.4.1 Overview of ways to generate ontology mappings

Ontology mappings are used for facilitating interoperability between ontologies, and they connect the meaning of a concept in one ontology with the meaning of a concept in another ontology. Mappings between two concepts are the most important ones.

There is research that is focusing on inferring ontology mappings as its ultimate goal and (Kalfoglou, et al. 2003) provides a good overview. The basic underlying assumption of most approaches is that the ontologies are populated by instances. Then various forms of induction can be used to hypothesize about the mappings between a concept in one ontology and a concept in another one. These methods can also utilize the structure of the ontologies, the linguistic cues provided the concepts names as such and try to find a common ground for instances, e.g. by analysing text documents where these instances could be found. Therefore, ontology mappings can be said to originate from hypotheses that are created on the basis of this data. The output of these systems is often a value between 0 and 1 that express the strength of mapping between two concepts, i.e. how related they are to each other. Some systems can, however, instead return the logical relationship between these two concepts. In general, the purpose of these systems is to map, align or merge ontologies (and sometimes even database schema). We will mention more about two such interesting systems: CTXMATCH and MAFRA (see Bouquet et al., 2003b and Maedche, et al. 2002 respectively).

Another origin of ontology mappings is, again, human insight and knowledge. More precisely, it implies that a domain expert has knowledge of the domains of the two ontologies that are being mapped. As was the case with ontologies themselves, systems that generate ontology-mappings as output, could be integrated with our infrastructure mechanism and provide mapping proposals that are sent to it as input to be processed. However, let us now return to the discussion of systems that mechanically generate mappings.

2.1.4.2 Inferring ontology mappings using logic-based approaches

The purpose of CTXMATCH (Bouquet et al., 2003b) is to infer mappings between a concept in one hierarchical classification and a concept in another hierarchical classification. It infers one of five mappings types and these are related to the mapping types of C-OWL (see below) and related to our five mapping types (see §3.3.2). CTXMATCH uses a lexical dictionary (WordNet) as background knowledge in addition to logical background knowledge, and maps words to formal concepts, and does automated reasoning using algorithms for solving this as a satisfiability problem (i.e. a SAT solver) in order to infer the relationship between two concepts. Because they rely on linguistic knowledge, that linguistic knowledge must be fully correct with regard to the hierarchies that are matched, in order to for the system to guarantee correctness of the mappings proposed. Indeed, their algorithm focuses on applicability to various linguistic hierarchies, instead of an absolute notion of correctness. However, if we disregard the linguistic background knowledge, then their system as well as ours does logical reasoning about mappings and ontologies, but in different ways and in our case this reasoning is an ingredient in the whole infrastructure framework.

Research by (Giunchiglia et al., 2003) analyzes the semantic matching problem in depth and identifies two main approaches. The first approach is element-level semantic matching (that can either use weak semantics or strong semantics). The second approach is structure-level semantic matching. The first approach includes techniques that for example look for similarities in label names (weak semantics) or lexical word-meaning (strong semantics), whereas the second approach translates the problem to formal logic and solves it using reasoning. In our system we only use structure-level semantic matching.

2.1.4.3 C-OWL and it five mapping types

In our approach we focus on managing ontologies that have a formal logical meaning, i.e. every relationship has a formal logic meaning rather than a numerical value expressing similarity or dissimilarity. It is therefore interesting to investigate if there is any research about formalising the meaning of such mappings and deciding which types are interesting in the first place. Our source of inspiration comes from C-OWL and the five mappings that are proposed there (Bouquet et al., 2003). The authors describe a context-sensitive version of OWL, where ontologies are contextualised. They also mention the five bridge rules between different context spaces (equivalent, is subsumed by, subsumes, compatible and disjoint – presented in order below):

$$i: A \xrightarrow{\equiv} j: B, i: A \xrightarrow{\sqsubseteq} j: B, i: A \xrightarrow{\sqsupseteq} j: B, i: A \xrightarrow{\perp} j: B, i: A \xrightarrow{*} j: B$$

They also adhere to the tradition that every piece of knowledge has to be annotated with its context, and we also follow this convention. In contrast to the mapping language that we are using, their bridge rules are unidirectional.

2.1.4.4 Why ontology mappings can be intensional

In a philosophical paper (Majkic, 2005) it is argued that ontology mappings should be intentional rather than extensional. According to Majkic, the incorrect way to define mappings between concepts in two different knowledge systems is to do it in an extensional way, because then all the extensions (i.e. instances) of an ontologies in one of them will then automatically be propagated to the other system (what one system knows will imply what the other system knows). The correct way is to define an intensional mapping that only says that the meaning of the two concepts is the same (the instances of the mapping are not propagated). We haven't investigated the computational implication of this stance, because the current version of our infrastructure does not yet maintain instances, so we have not specified if instances would be propagated. However, the consistency that our infrastructure mechanism maintains is currently indeed at the ontology level and the mappings are intensional. It remains to be to investigated in the future how feasible it is to maintain consistency (using spheres of consistency) between various ontologies, mappings and instances that could be propagated over the mappings and ontologies. The author (Majkic, 2005) believes that intensional mappings could e.g. be useful in P2P systems, because then instances of certain concepts could rest with their ontologies in their nodes instead of automatically being propagated to other parts of the system.

2.1.5 How important is the logical language and its expressivity?

We now repeat some of the goals of the thesis and contrast this with the tradition prevailing in related research fields. One of the aims of this thesis is to investigate if and how semantic autonomy (as defined in chapter 1) is possible. This means that a certain formal knowledge representations is used to define several ontologies and evolving and linked with mappings (but in the future even more informal knowledge representations could be investigated). The important properties are semantic autonomy (as defined in chapter 1), semantic reliability (it is possible to define constraints that are maintained in order to facilitate interoperability and eliminate/reduce miscommunication) and scalability (the infrastructure mechanism can be used to maintain extensive constellations of connected and/or huge ontologies). In this paradigm where semantic autonomy, semantic reliability and scalability are important, the issue of expressivity of the knowledge representation is not primary but it should be subordinated the other infrastructure properties that are seen as fundamental. The correct question is then, given that we want to have semantic autonomy, semantic reliability and scalability, what logic can we then choose that fits with these requirements? It might be a logic that is not very expressive, but that has other good properties considering these requirements.

In chapter 3 we describe the particular logic we have chosen. The primary reason for choosing it is that it satisfies the requirements mentioned here (semantic autonomy, semantic reliability and scalability) and it e.g. has an explicit notion of state that is useful when performing ontology evolution. However, in the future we would like to see how this framework can be used with other logics that satisfy the mentioned requirements (see also chapter 8 about future work).
Description Logics (defined e.g. in Horrocks 1997) are a family of logics that are being advocated by a part of the semantic web community (e.g. Horrocks 1997 & 2002). Description logics can be expressive, but the price they pay for expressivity is high time-complexity. In contrast to the logic that we use currently, description logics typically use tableaux decision procedures for doing reasoning (Horrocks 1997), whereas our reasoning mechanism is creating a proof search tree.

2.1.6 Theories of Context

There are many theories about contexts and attempts to classify and formalise what context is. We will focus on the proposals from within the informatics, logic and artificial intelligence communities rather than on proposals from the fields of linguistics, cognitive science etc.

One interesting approach for classifying contexts is from (Guha, R. et al., 2003). The authors describe different kinds of contexts and how contexts can be used in different ways. More precisely, they define four different context types: Projection Contexts, Approximation Contexts, Ambiguity Contexts and Mental State contexts. They discuss *lifting relations* that can lift facts from one context to others.

- **Projection contexts** can be seen as specific situations or circumstances that are projections of more general situations or circumstances. This means that the specific contexts have some additional assumptions in comparison to the more general contexts. E.g. Cyc's[2] microtheories can be seen as such. As an example *WorkplaceContext* in Cyc describes some assumptions that are valid when a person is at work.
- **Approximation contexts** are contexts that create an approximation or abstraction of a more general context that has more fine-grained information. In contrast to projection contexts, approximation contexts create knowledge bases that can be inconsistent with the more general context that is approximated. They mention an example of a database with basic prices and another database where the prices also include shipping costs, spare parts, inventory costs etc. The first database can then be seen as an approximation of the second one.
- **Ambiguity contexts** preserve an ambiguity in the knowledge representation, but background knowledge can be used to resolve the ambiguity. One advantage of ambiguity contexts is that they facilitate briefer KR because not all underlying assumptions have to be made clear. E.g. they are used in natural language in *discourse context.* One examples is the use of indexicals in natural language, such *he, she, it.*
- **Mental State Contexts** describe either fictional contexts that contain fictional knowledge or are cognitive perspectives of reality.

[2] Cyc is multi-contextual knowledge base described at http://www.cyc.com

The reader can return to the following detailed comparison after having looked at chapters formalising our approach (chapters 3, 4, and 5). How is this related to our use of context when formalising or configuring a knowledge infrastructure? The closest classification from the list above is that of "mental state contexts" because we view contexts as being carriers of ontological knowledge that represent a viewpoint of reality and not of an exhaustive representation of reality. However, one could use our infrastructure mechanism to define sets of contexts that behave like projection or approximation contexts, by choosing the right set of mappings between one context and its projection or approximation context. In these cases, many of the mappings would probably be subsumption mappings. When simulating a projection context, a sphere of consistency could contain both the context and its projection context, considering they are supposed to be consistent. In the case of an approximation context, two different spheres of consistency could e.g. be used (these will be defined in §4.3).

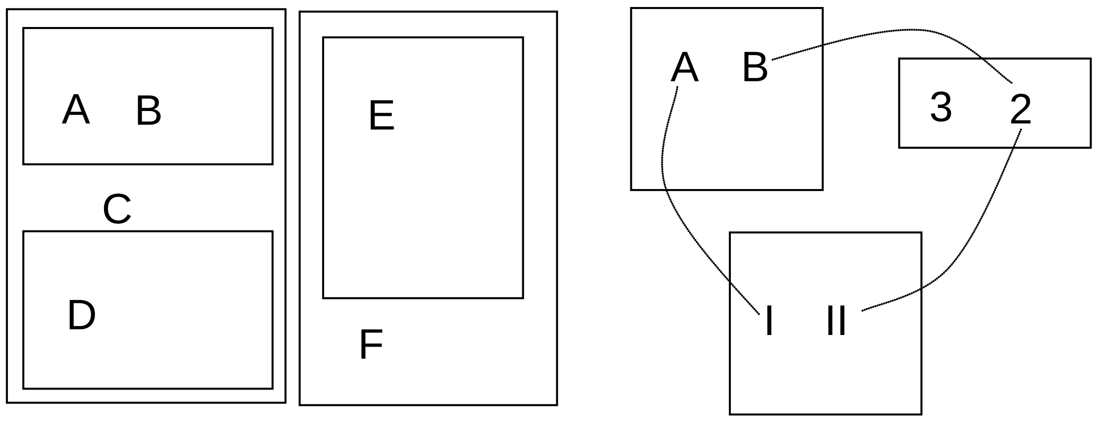


Figure 3. On the left side we illustrate an example of a divide-and-conquer context model, whereas on the right side we illustrate a compose-and-conquer context model. In the left model, reality can be cut and divided into pieces, and e.g. the knowledge fragments A and B belong to two a context and its super-context. In the right model, there is no global model nor global language, but only partial models having their own language. It is possible to create mappings between the different knowledge fragments that correspond to each other. These illustrations are inspired by similar ones in (Bouquet, et al. 2001).

2.1.6.1 Divide-and-Conquer vs. Compose-and-Conquer Context Models

Another classification of contexts that is even more relevant to our approach is the one by (Bouquet, et al. 2001). This envisions two main kinds of context theories in AI: divide-and-conquer and compose-and-conquer (see Figure 3).

According to this classification, divide-and-conquer sees a context as a way of partitioning (and giving a more articulated internal structure to) a global theory of the world. Compose-

and-conquer sees a context as a local theory, namely a partial, approximate representation of the world, in a network of relations with other local theories.

In compose-and-conquer context sensitivity:

- There is no general representation language, because it is context dependent.
- Denotation and truth are by definition contextual.
- Reasoning is done locally in the various contexts. Potentially, contexts could even use different reasoning rules.
- Relationships between contexts are not necessarily stated and used at a meta-level.

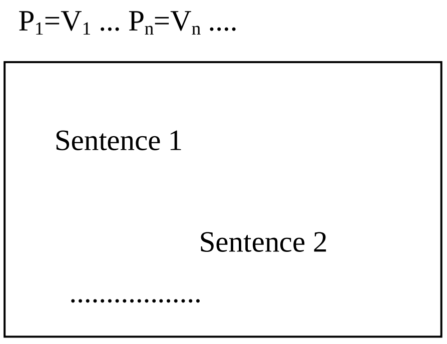


Figure 4. Context as box. The parameters and their values represent contextual dimensions and their contents. (copied from (Benerecetti, et al., 2000)).

Data Integration can be done using either divide-and –conquer or compose-and-conquer context sensitivity. In the first case, all data is mapped to a global schema whereas in the second case one relates semantically heterogeneous with each other. Our approach was inspired by compose-and-conquer context sensitivity so that is its intended default use, but it can be used for simulating divide-and-conquer context sensitivity as well.

2.1.6.2 Situations and Contexts

The final review of context theories we will mention is (Giunchiglia, 1993). According to this approach, a context *c* is a subset of the complete state of an individual that is used for reasoning about a given goal. The paper discusses how *situations* are related to *contexts*. One possibility is that a situation should be described using several contexts, each being an approximate theory of the same situation. The second possibility is to associate contexts to situations, and model time evolution of the world. The third possibility is to make one context to correspond to many situations. If we view situations as ordered states, then the formalism of our approach (see chapter 3) can be viewed as creating a grid that combines situations and contexts (e.g. the counter-models we present in Chapter 8 we have this grid-like structure).
According to the author (Giunchiglia, 1993), reasoning inside a context (see formal

definition below) could use different logics and/or different reasoning rules in every context. Therefore, every context could have its own reasoning engine. It is possible for a context to be an abstraction of another one. One possible definition of context the author above gives is that if L_i is a first-order language, Δ_i is the set of inference rules associated with a set of facts A_i, then he defines context c_i to be the triple $c_i=\langle L_i, A_i, \Delta_i\rangle$. They then describe bridge rules that connect facts in one context to facts in another context – these are inference rules.
In our approach we do not completely change inference rules in different contexts, but if contexts are put into separate spheres of consistency, then different variations of the same kind of reasoning is possible, because the level of consistency could be set to different values in different spheres (see section 4.3).

2.1.7 Distributed and contextual reasoning

Let us first review some interesting work about contextual reasoning by (Benerecetti, et al., 2000).
In their model they view context as a box (see figure 3). Inside the box there are some linguistic expressions about a domain, but outside the box there are some parameters with given values. The idea is that these contextual parameters and their values are used for fully determining the meaning of the expressions within the box.

They list three different contextual reasoning methods:

Localised reasoning – using knowledge from within a context when reasoning with that knowledge is enough.
Push and pop – contextualising or decontextualising knowledge. *Push* adds a contextual parameter to a knowledge base, whereas *pop* removes one such parameter. E.g. if we have logical statement that we want to make situation or time-independent, we would use *pop*.
Shifting – This occurs when we change the contextual parameters to new values, and therefore the knowledge represented has to be changed in order to account for this change. E.g. if the viewpoint is changed, the object or knowledge observed will look different.

They list three context dimensions that correspond to the three reasoning methods above.
Partiality. A representation only describes a subset of the state of affairs.
Approximation. An representation that abstracts away some aspects of the state of affairs.

Perspective. A representation that encodes spatio-temporal, logical and coginite point of view on a state of affairs.

The authors mention how these categories fit together. Localised reasoning can exploit a partial representation. Push and pop allows for varying the degree of approximation. Shifting can be used for changing perspective.

Our current framework supports localised reasoning and takes into account perspective. However, it does not have specific support using multiple context dimensions, because currently we only support one context dimension explicitly. A system for supporting several context dimensions would be more complex, and one could then use *push* for entering specific knowledge bases where all context dimensions have a value and *pop* for looking at the knowledge in a more general way.

2.1.7.1 Principles of Locality and Compatibility

The innovative paper by (Ghidini, et al., 2001) first describes two principles that should be used when formalizing context:

Principle 1. Locality. Reasoning uses only a part of what is potentially available (e.g. what is known, the available inference procedures). The parts being used while reasoning is what they call context (of reasoning).

Principle 2. Compatibility. There is compatibility among the reasoning performed in different contexts.

These principles have inspired our definition of semantic autonomy (see chapter 1).

2.1.7.2 Local Model Semantics

In (Ghidini, et al., 2001) Local Model Semantics are defined in the following way. $\{L_i\}_{i\in I}$ is a family of languages defined over a set of indexes I. L_i is a language used to describe what is true in a context. Let $\overline{M}_i$ be the class of all models (interpretations) of L_i. Every m$\in \overline{M}_i$ is called a local model (of L_i). Then they define what a *compatibility sequence* is and *compatibility relation.* These create constraints between the different local models. Then they a *model* for $\{L_i\}_{i \in I}$ in such a way that inconsistent context structures are excluded. Then they define satisfiability and validity. Satisfiability is defined for a given context. Local Model Semantics show some similarity to our formal definitions in chapter 3. However, the main difference is that our logic contains both contexts and states, so the languages *L* are given a two-dimensional index of both context and state.

2.1.7.3 Multi-Context Systems

Multi-Context Systems (Ghidini, et al., 2001) are a proof-theoretic analogy to Local Model Semantics. Assume that *I* is a set of indexes. A Multi-context system (MC System) is a pair

$$MS = \left\langle \{T_i\}, \Delta_{br} \right\rangle$$

where

- *for each* $i \in I$, $T_i = \langle L_i, \Omega_i, \Delta_i \rangle$ *is an axiomatic formal system where* L_i *is the language,* $\Omega_i \subseteq L_i$ *is the set of axioms, and* Δ_i *is the set of inference rules.*
- Δ_{br} *is a set of inference rules with premises and conclusions in different languages.*

They make the interesting distinction between Δ_i called *internal rules* and Δ_{br} called *bridge rules.* Internal rules have premises and conclusions within the same language, whereas bridge rules have premises and conclusions in different languages. They use this notation for these two kinds of rules (here *i* and *j* are context identifiers):

$$\frac{i:\phi_1 \quad \ldots \quad i:\phi_n}{i:\psi}\,ir \qquad\qquad \frac{i:\phi_1 \quad \ldots \quad i:\phi_n}{j:\psi}\,br$$

According to the authors, the principles of locality and compatibility are followed here, because both the language and inference rules can be local to a context, whereas the principle of compatibility is represented due to bridge rules that propagate reasoning across contexts. A complementary presentation of Multi-Context Systems can be found in (Bouquet, et. al, 1998).

Here follows a discussion that requires the reader to know the formal description of our system given in chapters 3 and 4, so the reader can return to this discussion later.
How is this related to our formalism? It could possibly be used for describing the reasoning in a simplified version of our system (we haven't include the state numbers here), e.g. by stating

$$\frac{i : COR(A_i, B_i) \quad ... \quad i : COR(B_i, C_i)}{i : COR(A_i, C_i)} ir$$

This rule means that the correspondence relationship is transitive when used inside an ontology. However, we see this relationship as something that is a logical consequence of the rewrite rules in sections 4.7-4.11. I.e. we see them as being primary, and they are applied in three phases, so it is not obvious one could express them using this formalism. Another challenge is that these rules are enabled by contexts only, whereas our reasoning rules also depend on states (see section 4.7). Finally, we allow for contexts that have ontologies connected by cyclic patterns of mappings. So it is important to rather specify the reasoning algorithm (see 4.12.3) because it defines in which order reasoning rules should be applied, and make sure that loops are prevented (considering that we allow for cycles of mappings). However, in future work (see chapter 8) it is possible that one could try to simplify the logical language and perhaps use reasoning rules more similar to these ones, but then still specify the reasoning algorithm using them, because it will specify in which order and when they are to be applied. Nevertheless, §8.2 is using a related notation for describing the reasoning rules in an abstract way.

2.1.7.4 Distributed Reasoning Systems

DRAGO is distributed reasoning system (Serafini, et al. 2005) that supports reasoning with several OWL ontologies (not more expressive than $SHIQ$ - this particular description logic is defined in Horrocks 1993) connected with bridge rules. This definition demonstrates which bridge rules that the system accepts:

A bridge rule from i to j is an expression of the following two forms:

1. $i : A \xrightarrow{\sqsupseteq} j : G$, onto-bridge rule

2. $i: A \xrightarrow{\sqsubseteq} j: G$, into-bridge rules

where A, B and G, H are concepts DL_i and DL_j respectively.

In their work DL_x refers to a description logic, and the DRAGO distributed reasoning system supports a collection of such logics that are connected by these bridge rules. They define a distributed tableau algorithm that creates a tableau for a regular description logic knowledge base, but when it reaches bridge rules, then the tableau reasoning does propagate into the connected knowledge bases. We like the idea that they extend standard reasoning with mapping propagation. The distributed T-Boxes have to be acyclic and be without individuals.

How is this compared with out framework?

- **Mappings that are unidirectional and subjective, vs. bidirectional and objective.** Firstly, their bridge rules are unidirectional and subjective in the sense that they belong to a single ontology and do not have an objective existence. This is more general than our choice of using bidirectional mappings, but considering our application we are not convinced that unidirectional subjective mappings are necessary from a practical point of view. In the definition of bridge rules above, the mappings only belong to ontology *j.* So ontology *i* could disagree about the existence of this mappings. It is true that they can be used for preventing inconsistencies to propagate, but we believe that using spheres of consistency for this purpose is more intuitive, because it is easier to visualize their impact on the ontologies stored compared to visualizing the consequences of overlapping mappings where reasoning can only propagate one way.

- **Should knowledge representation be separated from control over reasoning**? In our approach one can then separate representing mappings from controlling how far reasoning propagates and consistency is managed, and we think that is a convenient solution. In the future we would like to support changing spheres of consistency while the knowledge stays constant (see chapter 8) and then representation of knowledge and the control over how reasoning is done will be de-linked.

- **Mapping types supported.** In our system we can represent five different mapping types, whereas there are three in their system (equivalence is achieved by combining two bridge rules). Some of our mappings cannot be constructed from their mappings.

We should then say that their system supports more expressive ontologies than ours. Neither their reasoning system nor ours currently supports individuals. Finally, DRAGO is a reasoning system whereas the reasoning is only an element in the infrastructure mechanism, where modelling ontology evolution, control and change and decisions processes is equally important.

In the work by (Zhao, et al. 2007) they assume a scenario with several ontologies (so called "autonomous ontologies") and that one ontology can use concepts from other ontologies by a semantic binding. Two different tableaux methods are presented: one uses “cautious reasoning” that uses reasoning within an ontology and its neighbouring ontologies and “brave reasoning” that transitively includes an ontology’s neighbouring ontologies and their neighbouring ontologies etc. The reasoning algorithm they use builds a tableau and does satisfiability checking – this can e.g. be used for checking the subsumption relation between two concepts. It seems that "cautious reasoning" is similar to using a sphere of consistency containing a single ontology and enforcing full consistency, whereas "brave reasoning" corresponds to using global consistency from the point of view of an ontology. We believe there are (infinitely) many configurations of spheres of consistency that could not be represented using their formalism, that seem to have a different goal.
A similar comment can be made about this interesting work as above, namely that it is a reasoning system whereas the reasoning is only an element in the infrastructure mechanism, where modelling ontology evolution, control and change and decisions processes is equally important. Also, they have not included and integrated a notion of proof-bounded consistency in their framework.

2.1.8 Requirements for modular ontologies formalisms

The work by (Wang, et al., 2007) investigates semantic formalisms for modular ontologies. This investigation is done as a part of the NeOn project (see also §2.2.3). It does not at all investigate ontology evolution or process modelling in distributed systems. But they mention in general some properties that modular ontology systems should have:

Networking

- *Encapsulation* (support of using local theories)
- *Reusability/Inheritance* (the ability to import one ontology to another)
- *Authorization* (Controlling access to linked ontologies)

Dynamics

- *Networked Ontology Dynamics* (propagating changes and updates)

Distribution

- Loose Coupling (interconnections are well controlled and conflicts easily detected)
- Self-Containment (one can answer queries using the knowledge of one ontology)

Reasoning

- Complexity and Scalability
- Reasoning support for Terminological and Assertional Knowledge

Expressivity

They compare five different description-logic based formalisms for modular ontologies – all of them have better expressivity than our current logical language but at the cost of exponential worst-time time complexity. The authors mention the interesting distinction between two different approaches: 1) linking/mapping between ontologies and 2) importing (parts of) ontologies into other ontologies. We use the first approach.
Their research is focusing on other aspects than ours, and in this comparison the "networked ontology dynamics" had very poor support among the mentioned formalisms. Their main focus is on expressivity of the mappings language.

Our approach supports: encapsulation, networked ontology dynamics (but only a basic form due to few ontology operations currently), loose coupling, good complexity and scalability. Authorization is not the focus of our research. To summarise, we think that the emphasis of their comparison is to investigate expressivity and semantic meaning of formalisms for modular ontologies, whereas the ability to model dynamic aspects of change between several ontologies is not formalised at all (due to poor support of these features by the mentioned formalisms).

2.1.9 Para-consistent Logic and Model-theoretical Approaches for Computing an Inconsistency Degree

In the work by (Ma, et al. 2007) they develop a model-theoretical way of measuring how much inconsistency there is in an ontology. In this approach they define a four-valued logic where first-order theories have four-valued models. The set of truth values in this four-valued semantics is the following:

- *true,*
- *false,*
- *unknown* (or *undefined*) and
- *both* (or *overdefined, contradictory*).

In contrast to normal first-order logic (that maps each n-ary predicate to n-ary relation on the domain) a four-valued interpretation assigns a pair-wise n-ary relation $\langle P_+, P_- \rangle$ to each n-ary predicate *P*, where P_+ explicitly denotes the set of n-ary vectors which have the relation *P* under interpretation $\Im$ and P_- explicitly denotes the set of n-ary vectors which do not have the relation *P* under interpretation $\Im$. Based on these definitions, they define the truth value assignment to atomic predicates, and e.g. an assignment is both true and false if its n-ary relation both belongs to P_+ and P_- . The reason why we refer to their work is that they define an inconsistency degree that is model-theoretic. So the definition they give is the following.

Let Γ be a first-order theory and $\Im = (\Delta^{\Im}, \cdot^{\Im})$ be a four-valued model of Γ. The inconsistency degree of Γ w.r.t. to $\Im$, denoted $Inc_{\Im}(\Gamma)$ is a value in [0, 1] *calculated in the following way:*

$$Inc_{\Im}(\Gamma) = \frac{|ConflictTheo(\Im, \Gamma)|}{|GroundTheo(\Im, \Gamma)|}$$

Their inconsistency degree is the ratio of the number of conflicting atomic sentences (i.e. classified as being *both* true and false) divided by the amount of all possible atomic sentences formed from atomic predicates and individuals available. Later, they show a complex method and algorithm that reduces the amount of models that have to be investigated. However, even then their algorithm is only semi-decidable for first-order logic.

In some other of their other work (Ma, et al., 2007b) they describe the principles of how this method could be applied for calculating the inconsistency degree of an $\mathcal{ALC}$ Description Logic, but no algorithm or evaluation is yet provided. However in (Ma, et al., 2007c) they describe an algorithm that in polynomial time translates from the four-valued logic $\mathcal{ALC}4$ to $\mathcal{ALC}$ so that a traditional reasoner can be used. One of their challenges is that there are three types of implications in four valued-logic compared to one in normal description logic.

How do these approaches compare with our approach? Firstly, they focus purely on the reasoning or calculating a consistency degree, not on how to incrementally maintain a form of consistency and model change that maintains this consistency. Their consistency is model-based in contrast to our that is proof-based. Secondly, their approach falls into the paradigm of "Living in an Unmanaged World" that we analyse in §2.1.13 and the some of the conclusions of that analysis also apply to this work, e.g. the process of resolving inconsistencies can be indeterministic. In the case of four-valued logic, there is a certain freedom how to choose the implications types (consider there are three different ones) and different choices give different reasoning results. We therefore believe that one should be aware of the effort needed for resolving such an indeterministic situation into the "correct" one that corresponds to the intuitions of the users.

2.1.10 Belief revision and the AGM postulates

Belief revision has its roots in philosophy and investigates how belief systems should be revised. These belief systems can e.g. be viewed as belief sets that include not only axioms but also implicit knowledge, so they are closed under logical consequence. Instead of looking at particular algorithms for doing these revisions the research area rather investigates and formalises *properties* of such procedures. The three belief changes types are: *expansion*, *revision* and *contraction.* Expansion adds a new sentence ϕ to a belief system K together with the logical consequences of the addition. Revision adds a sentence ϕ that is inconsistent with K in such a way that K can accommodate ϕ.

2.1.10.1 How to compare the infrastructure mechanism with belief revision formalisms

It is not straightforward to compare our infrastructure mechanism with belief revision. However, the first approach (called approach 1) is to present abstractly what processing the infrastructure mechanism does, considering a given system K (of mapped ontologies) and a proposal ϕ as "inputs" and the resulting system K´ as "output". In this approach the ability of contexts to agree or disagree with a proposed change is included in this single operation step *op* (seen from an abstract point of view). We can therefore use these defined symbols and write

$$K \; op \; \phi = K'$$

This is the notation used by the belief revision community and *op* is normally one of the three following standard operators: *expansion, revision* or *contraction.*

We want to emphasize a particular aspect of our system, namely that even if there is no conflict (i.e. inconsistency) between K and ϕ then the infrastructure can still either accept or reject a proposals to add ϕ, because it has to be taken into account if a context or set of contexts accepts the change. We can express this particular case formally in this way:

$$K \nvDash \neg\phi \Rightarrow (K \; op \; \phi = K) \vee (K \; op \; \phi = K + \phi)$$

If we look at the definitions of expansion, revision and contraction we will learn that neither of these three operators actually satisfies the equation above, i.e. *op* can not be one of these three operators. So belief revision is a related theory, but does not provide a ready solution for our infrastructure mechanism.
We will also mention a second way of comparing belief revision with our infrastructure mechanism and in this second comparison does not include the contexts' process of accepting or rejecting in the operator *op.* Instead, in this approach it is assumed that we are in the case where ϕ is judged to be currently accepted (in other words: newly proposed knowledge is better than existing knowledge). We will call this approach 2.
The next section will investigate if our infrastructure mechanism satisfies the AGM postulates, when using approach 1 vs. approach 2 as a comparison paradigm.

2.1.10.2 Investigating if the infrastructure mechanisms satisfies the AGM postulates

In the belief revision community the so called AGM postulates have an important role (Alchourrón, Gärdenfors, and Makinson 1985). Therefore, we will look at the AGM postulates for revision. It is assumed here that this belief system is using belief sets as described above.

$\dot{+}$ is a function representing revision taking a belief set and a sentence as arguments and giving a belief set as a result. The belief system that results from expanding K by sentence ϕ will be denoted $K+\phi$.

$(K\dot{+}1)$	For any sentence ϕ and any belief set K, $K\dot{+}\phi$ is a belief set.
$(K\dot{+}2)$	$\phi \in K\dot{+}\phi$
$(K\dot{+}3)$	$K\dot{+}\phi \subseteq K+\phi$
$(K\dot{+}4)$	if $\neg\phi \notin K$, then $K+\phi \subseteq K\dot{+}\phi$
$(K\dot{+}5)$	$K\dot{+}\phi = K_{\perp}$ if and only if $\vdash \neg\phi$
$(K\dot{+}6)$	if $\vdash \phi \leftrightarrow \Psi$, then $K\dot{+}\phi = K\dot{+}\Psi$
$(K\dot{+}7)$	$K\dot{+}\,\phi \wedge \Psi \subseteq (K \dot{+} \phi) + \Psi$
$(K\dot{+}8)$	if $\neg\Psi \notin (K \dot{+} \phi)$, then $(K \dot{+} \phi) + \Psi \subseteq K \dot{+} \phi \wedge \Psi$

$(K\dot{+}2)$ says that input sentence ϕ is accepted in $K\dot{+}\phi$. $(K\dot{+}5)$ says that $K\dot{+}\phi$ should be consistent, unless ϕ is logically impossible. $(K\dot{+}6)$ says that logically identical sentences should lead to identical revisions. $(K\dot{+}7)$ says that revising K with $\phi \wedge \Psi$ is a subset of revising K with ϕ first and then with Ψ.

We now compare these postulates with the behaviour of our system as described in chapter 5. We will assume that revising K with ϕ corresponds to sending a proposal to our infrastructure mechanism that ϕ should be added, when the existing knowledge of the whole system is K (e.g. we assume there is one sphere of consistency and it contains K and we simplify the discussion by assuming that the system maintains full consistency within that sphere of consistency). Our infrastructure mechanisms satisfies $(K\dot{+}1)$ because the result of the processing is a new connected network of ontologies (note also that processing a proposal takes finite time - see §8.3). This is true for approach 1 and 2.

$(K\dot{+}2)$ above assumes that ϕ is a good piece of knowledge that should be added to K (using their language, "ϕ is accepted"). Our formalism has very different assumptions, because it first investigates if ϕ is consistent or inconsistent and then also asks the involved contexts if the change is desirable in the first place. i.e. we do not assume a-priori that the new knowledge is better than existing one, or is desirable at all. Even if it is consistent with K, it can still be decided to be rejected (i.e. we consider approach 1). But if we assume that ϕ is desirable by the contexts (i.e. considering approach 2), then our formalisms satisfies ($K\dot{+}2$). In the current prototype, however, we decided to not focus on belief revision due to its complexity, so in the current prototype if ϕ is consistent with K and ϕ is desirable by the contexts, then ($K\dot{+}2$) holds.

The formalisation of our infrastructure mechanisms satisfies ($K\dot{+}3$) no matter what happen. E.g. if ϕ is deemed to undesirable by the involved contexts, then $K\dot{+}\phi=K$, and even then the equation ($K\dot{+}3$) holds. For the same reason, we infer that ($K\dot{+}4$) does not hold, because if ϕ is not desirable by the contexts then ($K\dot{+}4$) leads to $K+\phi \subseteq K$ which is not true generally. Indeed, if the policy is that redundancy is forbidden then even if $\phi \in K$ and ϕ is desirable by the contexts, ($K\dot{+}4$) still does not hold because we again get $K+\phi \subseteq K$. However, if contexts accept all proposal that do not cause conflict (i.e. approach 2) then ($K\dot{+}4$) holds.

If we assume that we maintain full consistency then ($K\dot{+}5$) does not hold because the system is guaranteed to never lead to inconsistency. However, if the sphere of consistency is using bounded reasoning and the proposed ϕ creates a contradiction that is not worse than what is permitted and ϕ is desirable by the involved by contexts, then ($K\dot{+}5$) holds. In the current logic, it is not possible to have a single piece of knowledge that is self-contradiction (but one piece of knowledge can contradict another one).

($K\dot{+}6$) is an interesting postulate and we think that it illustrates one of the differences between belief revision and the infrastructure formalism. If we consider comparison approach 1 (as define above), then there could be a situation in principle where contexts agree to add a piece of knowledge ϕ because it corresponds to their intuition and conceptualization (i.e. the cognitive distance is small) whereas an equivalent piece of knowledge Ψ could express in a way that is impractical, unnatural or unintuitive and therefore it could be rationally rejected on these pragmatic grounds. Then ($K\dot{+}6$) does not hold. However, using approach 2 and disregarding this decision procedure of the contexts, then ($K\dot{+}6$) does hold, because the reasoning as such is only concerned by the meaning of these pieces of knowledge whereas in approach 1 it is the "human" input in the decision process, that on rational ground can and should be concerned with the form of the knowledge added, that would break ($K\dot{+}6$).

For a similar reason, it is clear that (K$\dot{+}$7) does not hold in approach 1. E.g. the contexts could prefer to accept $\phi \wedge \Psi$ as a single piece of knowledge instead of accepting ϕ and Ψ separately because 1) $\phi \wedge \Psi$ could actually correspond to their cognitive model whereas ϕ and Ψ separately would only do so imperfectly, or 2) because the contexts want a knowledge representation of minimal length, so they would rather accept a single piece of knowledge rather than two pieces of knowledge representing the same meaning. Using approach 2 for comparison will however accept (K$\dot{+}$7). Finally, (K$\dot{+}$8) does not hold considering approach 1, for similar reasons as mentioned (but this time the contexts could have a preference for many pieces of knowledge). But if we use approach 2, then (K$\dot{+}$8) holds. Let us now summarize our results.

2.1.10.3 A summary of the comparison

Here is a summary of our investigation:

AGM Postulate	**Does the knowledge infrastructure formalism satisfy the given postulate?**	
	Considering Approach 1	Considering Approach 2
(K$\dot{+}$1).	Yes	Yes
(K$\dot{+}$2).	No	Yes
(K$\dot{+}$3).	Yes	Yes
(K$\dot{+}$4).	No	Yes
(K$\dot{+}$5).	No (assuming p_c=1)	No (assuming p_c=1)
(K$\dot{+}$6).	No	Yes
(K$\dot{+}$7).	No	Yes
(K$\dot{+}$8).	No	Yes

We believe that approach 1 is a more fair comparison because it takes into account the "pragmatics" of evolving knowledge where contexts express their opinions about a change. But we have also presented approach 2.

2.1.11 Truth maintenance

Truth maintenance is mentioned as a traditional paradigm that had its major impact in the past, and we will briefly mention the assumption-based truth maintenance system by de

Kleer (de Kleer, 1986). The important assumption of this system is the keeps track of justifications of knowledge and therefore how different facts are dependent on each other. The knowledge models variable assignments and rules that propagate variable assignments to other variables. When a rule is added that creates a contradiction, the system finds consistent subsets that can be used to derive data.

This is very different from our approach, because our system does not maintain a hierarchy of justifications of earlier knowledge. In order to simplify things, our system does not do this, because the added knowledge very seldom follows from existing knowledge. Indeed, if it does, then it creates redundancy, and some policies might forbid redundancy all together. However, the current logic does keep information about in which state a piece of knowledge was added, because that is a natural part of the logical formalism. In the future, when instances are supported, then also information about which instances are true for which predicates. Finally, in our approach we do not try to proactively partition facts into consistent subsets when a contradiction occurs. Instead, we define in advance sets of ontological relations where a certain level of consistency has to be maintained. This is different from the partitioning in truth maintenance where dependencies of justification are maintained.

2.1.12 Process modelling using a formal protocol

In chapter 5 we describe a protocol language that formalizes the computational behaviour of the knowledge infrastructure. Our language was inspired by LCC (see Robertson 2004), but then it has become rather different in some respects. LCC is a generic language for modelling a interaction protocols for a Multi-Agent System and it uses explicit message passing as a means of communication between different agents. Our formalism does not use message passing, but instead it uses rules that connect the behaviour of the various entities, in our case contexts and infrastructure mechanism. It would be possible to re-write our rules to a protocol version that uses message passing, but for the purpose of this thesis, we were not convinced about the benefit of this. Of course, when the different entities are physically distributed some form of underlying communication is needed for sending information between them and one should then decide if this is done implicitly of the rule-interpreter or explicitly in the specification language. We have done it implicitly.

2.1.13 The alternative: living in an unmanaged world

In the excellent work by (Huang et al., 2005) the authors investigate a paradigm where inconsistencies of arbitrary depth or type are allowed to exist in a single ontology, but when a query is sent to such an ontology then a meaningful answer is extracted (this is formally defined). The main idea of their approach is to automatically choose a consistent sub-theory and use a traditional query-answering method for receiving an answer from that sub-theory. They give an interesting example of an inconsistency in which a brain is considered to be both a body part and a central nervous system, whereas body parts and nervous systems are considered to be disjoint. Here are some of their interesting definitions:

Terminology:

ϕ is query

$\sum$ is an ontology

$\vDash$ is the traditional consequence relation

$|\approx$ is the consequence of an inconsistency reasoner

Definitions:

Definition 2. An inconsistency reasoner $|\approx$ is sound if

$\sum |\approx \phi \Rightarrow \left(\exists \acute{\Sigma} \subseteq \Sigma\right)(\Sigma \nvDash \perp \wedge \acute{\Sigma} \vDash \phi)$

According to this definition the inconsistency reasoner is sound if there is a consistent subset that would have returned an answer (using a regular reasoner) that is the same as the answer returned by the inconsistency reasoner.

Definition 3. An answer given by an inconsistency reasoner is meaningful iff it is sound (as defined above) and consistent according to this definition:

$\sum |\approx \phi \Rightarrow \sum |\not\approx \neg \phi$

They then describe two selection functions that decide which consistent subset to choose. The first one starts from nothing and gradually grows a consistent knowledge base by adding new axioms to the existing consistent subset. The other selection function starts with the whole ontology and gradually removes axioms until the result is consistent. Both these approaches are called linear extension approaches. One problem with both these approaches is that they are indeterministic and it is difficult to say if they have found the "right" consistent subset. They then define a method for the selection formulas to choose new axioms (that will be added to the consistent answer set) that are "relevant" to the query. The describe a procedure that uses backtracking when it has done incorrect choices and they call it *over-determined processing* (ODP).

The authors give the following computational complexity result for ODP. The complexity of the over-determined processing is $n^k \cdot C$, where n is $|\sum|$ and k is n - $|S|$ (S is the largest consistent subset in $|\sum|$ and C is the complexity setting). Their experimental evaluation shows good results, but they conclude that the algorithm still cannot for certain choose the "right" consistent subset.

Let us now compare this paradigm of not managing ontologies with our proposed paradigm of having an infrastructure that manages the ontologies. Firstly, the approach described here is for a single ontology and not explicitly for a network of connected ontologies. Secondly, even if we disregard the notion of consistency, their model does not formalise how change happens (in ontologies and mappings connecting them). In our infrastructure mechanism, one can create a big sphere of consistency where p_c=2 and then consistency checking is turned off, but when mappings are proposed to be added or deleted then the infrastructure still make sure the changes are wanted by the involved ontologies. In that situation, when some inconsistency is allowed, their approach is complementary and could be used for query-answering (that is not at all the focus of this thesis).

Thirdly, we reach the fundamental issue in the comparison. In their approach change to an ontology is uncontrolled. The benefit is that it saves computational effort during those moments when the changes happen. The drawbacks are the following:

- In a network of connected ontologies, there is no guarantee that the change is wanted (i.e. an issue orthogonal to consistency) because the social aspect of changing the knowledge are not modelled.

- When a person by accident adds a piece of knowledge that contradicts a piece of knowledge added by another person, there is no alert about this.

- If an infrastructure mechanisms keeps information about the consistency constraints it has satisfied so far, it can do less computations when somebody proposes to change its knowledge state (e.g. add knowledge) because it only has to incrementally investigate how this change affects the current "good" state, instead of having to investigate how *any* combination of knowledge elements could create a conflict that violates the assumed constraints. A "good" state can mean both full or proof-bounded consistency.

- During query-answering time it is very difficult to know which consistent subset that is the right one, because there is no mechanism that tracks when the system entered an inconsistent state.

- Query-answering can require substantial computational effort (as quoted above).

2.1.14 A P2P information system relying on redundancy

SWAP (Broekstra, et al. 2003) is a P2P system that supports distributed information sharing which consists of multiple peers that are connected with each other. In this system there are many pieces of knowledge annotated with meta-information, e.g. peer id, location, and additionDate fields – so some this information tells where the pieces of knowledge was created. It assumes that inconsistency and redundancy are very natural. It supports simple concept hierarchies that are extracted from underlying data (e.g. databases), but peers can also send information to other peers. Information is given a confidence rating and it uses RDFS (see Antoniou et al. 2008 for definition) for representing knowledge If several peers send the same statement, then confidence ratings of that statement will increase. According to the authors the knowledge in their system "does not represent truth but rather a collection of opinions supported by different sources of information". They assume that frequently occurring opinions are more likely to be true.

We summarize here some of their assumptions:

- Pieces of knowledge can be added to a peer by being extracted from underlying data or by being communicated from other peers.
- The same or similar information should appear in the system many times. This redundancy occurs because peers could have replicas of information from other peers.

- A piece of correct information will occur more frequently in the system (e.g. is more redundant) than incorrect information.
- Inconsistency and redundancy of information are both desirable properties of the system, but the trust mechanism should help correct information to increase in confidence.

It is interesting to mention this system because it is also concerned with representing knowledge in a distributed environment but follows a radically different paradigm compared to our proposed infrastructure mechanism. We believe that there is a need for distributed systems that have a more statistical approach to representing information, but our reservation is that the foundations of such a system and its evaluation should be made clear. Therefore, we are not sure if ontology evolution happens in the SWAP system and if ontology mappings are represented at all. If not, then it does not support semantic autonomy (see chapter 1) because knowledge is only physically distributed but the semantics do not really evolve (either at all or in a manner where it is still possible to map meaning between different parts of the system). Secondly, it seems that the system is trying to deal with these three deep problems at the same time:

1. **Disinformation** – two pieces of knowledge are different because one is wrong (on purpose or by accident).
2. **Subjective Knowledge** – two pieces of knowledge are different because they both express knowledge about subjective states of affairs, so they do not even talk about the same domain (i.e. the domains only belong to individual user and not to a shared reality).
3. **Cognitive Context** – two pieces of knowledge are different because they use languages belonging to two different cognitive models of the same reality.

We cannot see how the proposed system is able to deal with these problems and distinguish them from each other. E.g. when calculating confidence rating, then some form of majority vote could possibly work for 1) if various peers have similar information access. But if 1), 2) and 3) are present at the same time that method won't work at all. More generally, we haven't seen anyone proposing a solution for how to deal with all of these at once.
Our proposed infrastructure mechanism deals with 3) and to some extent 1) (but in a non-statistical way, i.e. every piece of knowledge is seen as being true or false and we assume that most proposals are true). And from the description above it should clear how our

infrastructure mechanism differs from the described system here. Most importantly, they do not facilitate semantic autonomy in a way where one can guarantee inter-operability by guaranteeing consistency between some of the peers.

2.2 Related work

In this section we will review work that is more closely related to our work than the background section, because now we will investigate ontology management infrastructures, autonomic computing and network science.

2.2.1 MAFRA - A framework for managing mappings

We will now explore a mapping framework called MAFRA – A Mapping FRAmerwork for distributed ontologies (Maedche, et al. 2002). This framework describes a system where mappings evolution is done between ontologies. The focus is on providing a rich and expressive mapping language. Their system has a module that calculates similarity between an entity in one ontology and an entity in another ontology. This similarity is then used as input for creating mappings (called bridges), based on certain heuristics. Sometimes, new mappings are created based on similarity, heuristics and existing mappings. The established mappings are used for translating instances from one ontology to another ontology, either in off-line or on-line mode. In off-line mode the transformation happens once, whereas it happens continually in on-line mode. They also have a cooperative consensus building feature for establishing consensus on semantic bridges, but formalization of it is provided. The system can use background knowledge when proposing new mappings. Mappings can be done not only between concepts but also relations, properties etc. There is support of 1: n and m : 1 mappings. Every mapping is linked to a transformation procedure for transforming instances. There is also a hierarchy between the mappings and a mapping can specialize another mapping.

The main difference to our work is that their lack of process modelling – instead they focus on providing rich semantics for a variety of mappings. It is not clear from the paper if the system can reason with these complex mappings, and what the effort of that reasoning is. However, our system could adopt some of their improvements, e.g. 1: n and m : 1 mappings, because then it can be used in more complex and realistic scenarios. But then the computational complexity of that has to be investigated. They write that MAFRA is implemented to work with KAON, so it seems that MAFRA becomes a particular piece of software. This is a contrast to our work where the infrastructure mechanism specification could be implemented in various pieces of software, due to its transparent specification.

2.2.2 An infrastructure for reusing and evolving reused ontologies

The work by (Maedche, et al. 2003, 2003b) describes an infrastructure based on KAON that manages multiple ontologies. The main assumption of the authors is that ontologies should be *reused*, i.e. replicated and inserted into other ontologies, and that this dependency information has to be maintained, in order to maintain a special form of consistency, that actually means perfect identity between the original ontology and its replicas. They claim that ontology-based systems are a special case of software systems, and that because software-encapsulation and reuse is a good practice then also ontologies should be encapsulated in reused. We disagree with this approach, because these ontologies can consequently not be used for describing different points of view. On the contrary, all the ontologies that exist must be able to be combined in a conflict-free way, so they assumed a global knowledge model. But we still describe their approach.

Their infrastructure includes a centralised ontology registry, a way of reusing “distributed” ontologies and a method for evolution of distributed ontologies. They describe that a certain ontology A can *include* a certain version of ontology B (it must be the whole ontology – not a part of it). When one has imported an ontology into another, a dependency is created. They distinguish between

- **Single Ontology Evolution** – a single ontology evolves
- **Dependent Ontology Evolution** – several ontologies located on the same physical node evolve
- **Distributed Ontology Evolution** – several ontologies located on different physical nodes evolve

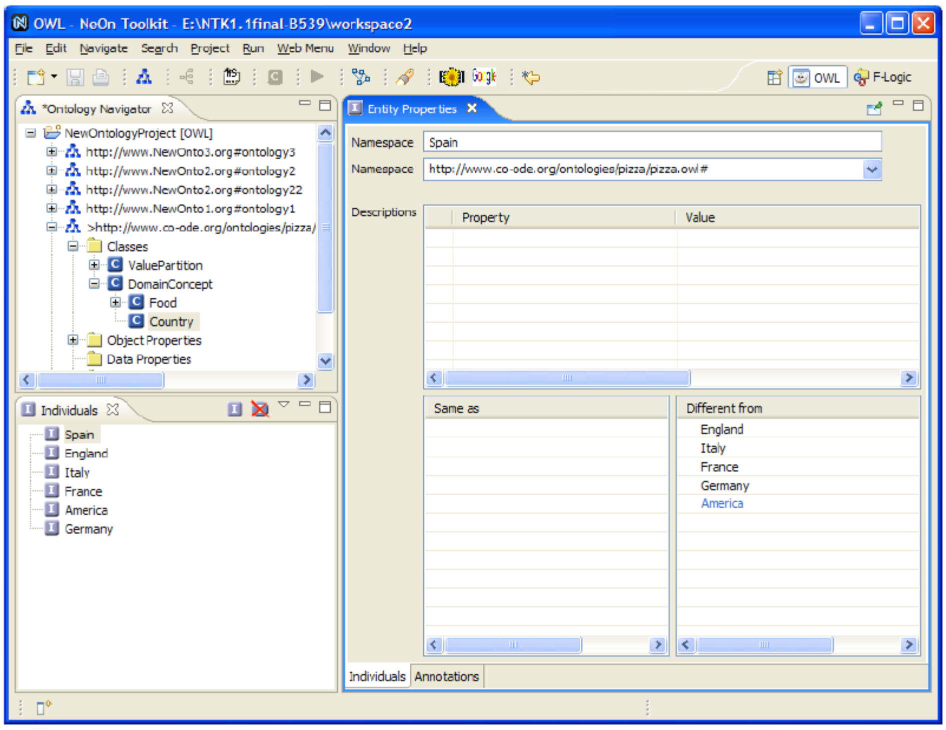
These ontologies are linked by inclusion links and that means in this context that an ontology A is re-using the whole of ontology B. In the case of dependent ontology evolution, their system uses the "push" paradigm to immediately propagate changes. So if a concept changes in ontology B then that changes is automatically propagated to ontology A. This means that they assume that there is a big value in making sure that all copies of an ontology that are included in other ontologies, must kept exactly identical to the original. We do not understand why this is the case, and our infrastructure is based on different assumptions.

In the third case above, their system creates "replicas" of the original ontologies. Replicated ontologies can not be modified either – instead the source has to be modified. Because there are dependencies, changes have to be propagated immediately using the “push” strategy, when they occur in one physical node. However, they are propagated using "pull" strategy across different physical nodes. This means in practice that a physical node has to request up-to-date information and this then dependent ontologies send so called "deltas" between replicas and the original ontologies. They define an ontology meta-ontology (OMO) for describing information about the ontology. Their infrastructure provides a mechanism for searching for other ontologies, and the matching is done using WordNet. For each concept a set of synsets is obtained.

From our point of view, the problem with their approach is that they do not allow for semantic autonomy (as defined in chapter 1), and they assume that ontology replicas have to remain *identical* to the originals, so the other ontologies cannot fully express a different point of view. According to the authors, they currently cannot integrate semantically heterogeneous ontologies.
Our framework mechanism currently does not support creating inclusion links that remain as dependencies between ontologies. However, we do not feel this is essential, but that it rather limits the semantic autonomy of the ontologies, because they cannot freely adapt the ontologies to their local needs – some of these adaptations will be automatically propagated to other ontologies even if they are not beneficial there. Using our formalism we rather the approach of making a copy of an ontology to another contexts and creating individual correspondence links between every element of the original ontology and the copied ontology. In this way, the freedom to change the ontologies is still great while coherence is maintained.

2.2.3 The Neon Project and the NeOn Toolkit

NeOn is an ambitious applied project that aims to “advance the state of the art in using ontologies for large-scale semantic applications in the distributed organizations”. The NeOn toolkit is a piece of software (a so called "Ontology Engineering Environment") that makes it possible to develop networked ontologies. Due to the size and complexity of the project, it


OWL - NeOn Toolkit - E:\NTK1.1final.B539\workspace2
File Edit Navigate Search Project Run Web Menu Window Help
OWL
F-Logic
*Ontology Navigator
Entity Properties
Individuals
Annotations

NewOntologyProject [OWL]
http://www.NewOnto3.org#ontology3
http://www.NewOnto2.org#ontology2
http://www.NewOnto2.org#ontology22
http://www.NewOnto1.org#ontology1
>http://www.co-ode.org/ontologies/pizza/
Classes
ValuePartition
DomainConcept
Food
Country
Object Properties
Data Properties

Individuals: Spain, England, Italy, France, America, Germany

Namespace: Spain
Namespace: http://www.co-ode.org/ontologies/pizza/pizza.owl#

Property	Value

Same as	Different from
	England
	Italy
	France
	Germany
	America

Figure 5. The NeOn Toolkit software, where users can edit ontologies using a graphical user interface.

also investigates some infrastructure components that we will not investigate at all, e.g. ontology visualization, authorization, complex-query answering, human factors in ontology design GUI and software tool design etc. The Neon Toolkit supports

- Basic schema editing
- Schema visualization and browsing
- Import and export of F-logic, subsets of RDFS and OWL

There are then commercial plug-ins that provide support for: rule support, mediation, database integration and queries. So it is a commercial project that makes the community dependent on their particular software.

The main difference between our infrastructure mechanism the Neon Project is that it is delivering a piece of software (the NeOn toolkit), and does not contain a precise formalisation like ours (see chapter 5) of its dynamic behaviour. It is therefore difficult to learn from their approach and use it for achieving semantic interoperability of the semantic web as a whole and its various semantic applications or for making other infrastructure components in the future (unless they become NeOn plug-ins). The NeOn toolkit has a plugin architecture, and we think this makes other solutions overly dependent on the Neon toolkit software instead of being more independent. It does not provide support for maintaining spheres of consistency as we have defined them.

2.2.4 Ontology-based Autonomic Computing Systems

In the outstanding research by (Stojanovic et al. 2004) they describe an ontology-based autonomic computing system that is developed for the purpose of automating system management. The focus is to develop better "correlation engines" and investigate if they benefit from using ontologies. Correlation Engines are defined as "autonomic core components that perform continuous automated analysis of enterprise-wide, normalized, real-time event data based on user-defined configurable rules". There is some similarity between this definition and our infrastructure mechanism definition. Autonomic computing systems monitor and gather data they need to react upon, according to their management tasks and targets. These systems are driven by events. Our infrastructure mechanism does therefore specify an autonomic computing system.

Their system is using a reference model for correlation engines and it has a three layers:

1. A resource layer
2. An event layer
3. A rule layer

The resource layer is using an object-oriented data model where a certain resource can be an instance of a more general resource concept. Events are special messages that indicate a change of state of a resource, e.g. component failure. There are two kinds of rules: correlation rules and action rules. Correlation rules can e.g. detect a form of system failure or suspicious network behaviour that occurs over longer time. Action rules trigger automatic remedy actions or gather additional monitoring data.

They mention how the eAutomation engine is improved so that it uses an ontology and a set of rules, instead of relying on hidden and hard-coded knowledge. It focuses on availability management of IT resources. E.g. its resource model has an attribute that describe in which state a resource is and which is the desirable state of the resource. There are two kinds of relationships between the resources, e.g. the first is a start/stop relationship that describe dependencies between resources. The eAutomation engine combines correlation and action rules into three types of rules: Correlation like "WHEN condition, THEN action", relationship correlation rules (e.g. that a resource can start after another has started), and request propagation rules (e.g. that there is a start request for a resource A, then it is propagated to components with which A has a *startAfter* relationship.

They mention the following advantages of ontology-based correlation engines: reusability, extensibility (e.g. the set of attributes can be extended), applicability (use ontologies to help with search of particular services), verification, integration, evolution, visualization, open standards. Also there are runtime benefits: justification, ranking and gap analysis.

Of all the related research we have investigated we think this piece of research is the most impressive considering the theory and applicability presented. It is similar to our research because we both describe ontology-based autonomic computing systems. However, their system is a particular application focusing on IT availability management and not a specification of infrastructure mechanism for facilitating evolution of ontologies and maintaining their inter-operability.

2.2.5 Network Science and Simulated Knowledge

Considering the infrastructure mechanism creates a network of connected ontologies, we briefly note that similar issues are investigated in network science. A particular approach that applies network science principles to knowledge simulation is explained in (Halladay et al., 2004) in a rather non-technical way. They argue that knowledge simulation is more graph-centric than knowledge representation and it does not have to human-understandable. *Simulated knowledge* is the field of study that deals with storing and accessing knowledge using approaches with a reliance on network science principles. *Network science* is a field of study that observes and models network-related phenomena.

They describe that a network of knowledge gives meaning to terms by means of connecting them to other terms. Therefore a "vivid meaning" is created by relationships' richness. However, humans do not need to be able to comprehend such a whole network, because they might not be able to grasp it, but they could be able to understand *parts* of it. They mention that the Internet as a whole is such an example. Also, it is mentioned that represented knowledge looks similar to natural language, but that is a danger, because e.g. knowledge elements should not be ambiguous whereas natural language is ambiguous. Indeed, that is why our simple ontologies consist of concepts that correspond to single meanings.

In the new paradigm they propose, knowledge should be inter-connected in a network form and they write that the biggest challenge is then to decide how to connect, automatically, the relationships with the conceptual network. This view is rather similar to the Semantic Web vision. Simulated knowledge expects relationship connections to become complex beyond human comprehension. This is also true for the network of connected ontologies that our infrastructure mechanism is creating. They advocate that knowledge should be viewed as a network rather than logic, but we think these two aspects can be combined if the knowledge representation is kept simple. It is one of the reasons we have kept it simple.

In network science sometimes it is not possible to actually replicate a real network and therefore one has to do simulations. The researcher Duncan Watts created networks by randomly connecting nodes and analysing the effect of these connections. Another researcher (László Barabási) then discovered that the amount of connections per node had a normal distribution, whereas in nature networks have a power-law distribution and are called scale-free networks. One explanation could be that older nodes have a greater opportunity to be the target of connections. They also mention that network science predict that a form of *phase transition* will occur network when the connectivity grows and various nodes can directly or indirectly via links influence other nodes. This is related to our experiments in §7.3 where we also observe a phase transition (by changing consistency level for a network, but not by changing connectivity level for a network). In our case the "influence" between the nodes actually corresponds to consistency that is more or less proof-bound. They write that "At the critical point, the set of nodes are no longer considered as independent nodes, but instead, the set of nodes become a cohesive network". Our framework mechanism maintains a form of network cohesiveness by maintaining consistency, or more precisely it is a cohesiveness of the knowledge represented by the network.

Chapter 3 A logical formalism for simple ontologies and ontology mappings.

The purpose of this chapter is to define the syntax and meaning of relationships within ontologies and mappings between ontologies. An ontology mapping is a relationship between a concept in an ontology and a concept in another ontology. The simple ontologies we describe have an expressive power that partially overlaps with a small subset of OWL as will be explained.

We will define the syntax of our ontology language and ontology mapping language and define their formal meaning.

3.1 Introduction to the underlying logical formalisation

We will define the syntax of our ontology language and ontology mapping language, but first we will briefly mention something about the underlying logical language that will be used to define the meaning of the syntax.

One characteristic of our general logical formalism is that there are several domains ($D_{i,m}$), and they have two indexes: one for describing the state and one for describing the local context. The reason for this is that our framework and system deals with a scenario where there are several contexts that have evolving ontologies – i.e. the state index formalises the notion of the system "moving forward" and changing. The properties and this discrete time will be formalised. We are using the notion of a concept *P* in an ontology *j* and that is denoted by the predicate $P_j()$ in the formalism. The logical formalization of ontology relations and mappings (below) makes the use of quantifiers that quantify over elements in a domain, but we will focus on how these ontology relations and mappings interact with each other, instead of focusing at the interaction between actual instances and ontology relationships or mappings.

3.1.1 The notion of context

Informally, contexts model several cognitive points of view of a reality because every context hosts an ontology that represents a model of such a view.

From a logical point of view a “context” is something that adds an index to the logical language, domain and interpretation functions (i.e. these are multiplied), so every context will have its own truth valuation function.
Also, it prohibits direct import of and access to concepts from other ontologies, because only ontology mappings can mediate such access.
As a shortcut in this thesis, when we say *ontology i*, that actually refers to the ontology of context *i*. The variable *i* is the then the context identifier, i.e. a given value of *i* refers to a unique context.

3.2 The epistemological assumptions – a motivation of the logical formalisation

The epistemological assumptions are mentioned because they form a specification that will be satisfied by the logical formalism. The epistemological assumptions are that

- There is an objective notion of what is true (but it is not expressible directly and as we will see it is only potentially and not always permanently accessible).
- There are several points of view that only express fragments of the objective notion, each in its own language.
- Both the notion of what is true (in a given state in the domain model that is independent of the points of view but inexpressible directly) and what the points of view can see can change.

The first assumption implies that the logical domains of all local contexts (in a given state) are subsets of one bigger domain. This relates to our original motivation of building a distributed knowledge management system for an organization – the fact that the organization exists in reality and exists as an entity is an argument for declaring that there is an objective (but deep and underlying, i.e. not obvious at the surface level – the level of the concept names used in different ontologies) notion of what is true.
However, the vocabulary is never allowed to utilize that big domain directly, so thinking again about the original motivation we can say that the unity of the organization is not expressed at the surface level (more concretely, the concepts the different divisions are using).

Instead, the vocabulary is only connected to the localized domains (this is the second assumption) so from the application point of view one can say that only the divisions have their own languages, not the whole organization as such.
The third assumption explains why in the logical formalization there is a domain model that has two indexes (for expressing both point of view and state). The interpretation from the application point of view is that the organization is changing and evolving, due to a stream of business needs.

3.3 Syntax of the languages

3.3.1 Syntax of the ontology language

We will confine ourselves in the presentation of sections 3.3 and 3.4 to an ontology language and ontology mapping language for which we will present efficient and complete reasoning (in another chapter).
An ontology in context *i* can contain statements of any of these four types (where A_i and B_i are concepts that belong to context *i*).

1. COR(A_i, B_i)
2. IS (A_i, B_i)
3. IS2 (A_i, B_i)
4. DISJOINT (A_i, B_i)

These statements are actually relationships between concepts, but can also be seen as constraints *within* ontologies.

3.3.2 Syntax of the ontology mapping language

A concept A_i in ontology *i* and a concept B_j in ontology *j* can be connected by any of these five mappings

1. COR(A_i, B_j)
2. IS (A_i, B_j)
3. IS2 (A_i, B_j)
4. COMPATIBLE (A_i, B_j)

5. DISJOINT (A_i, B_j)

These statements are actually mappings between concepts, but can also be seen as constraints *between* ontologies.

3.4 Extended syntax of the languages

We will now look at a more extended syntax for the ontology language and ontology mapping language, and this more fain-grained syntax will later be needed for defining the semantics of these languages.

3.4.1 Extended syntax of the ontology language

An ontology in context *i* can contain statements of any of these four types (where A_i and B_i are concepts that belong to the ontology of context *i*, and *r* is a state r>0). The state operators $\mathbf{N_r}$ and $\mathbf{G_r}$ are defined in section 3.5.

Syntax	**Extended syntax**
COR(A_i, B_i)	$\boldsymbol{N_r}(\forall x_i((A_i(x_i)\wedge B_i(x_i))\vee(\neg A_i(x_i)\wedge\neg B_i(x_i))))\wedge$ $\mathbf{G_r}(\forall x_i((A_i(x_i)\wedge B_i(x_i))\vee(\neg A_i(x_i)\wedge\neg B_i(x_i))))$
IS (A_i, B_i)	$\boldsymbol{N_r}(\forall x_i(\neg A_i(x_i)\vee B_i(x_i)))\wedge\boldsymbol{G_r}(\forall x_i(\neg A_i(x_i)\vee B_i(x_i)))$
IS2 (A_i, B_i)	$\boldsymbol{N_r}(\forall x_i(A_i(x_i)\vee\neg B_i(x_i)))\wedge\boldsymbol{G_r}(\forall x_i(A_i(x_i)\vee\neg B_i(x_i)))$
DISJOINT (A_i, B_i)	$\boldsymbol{N_r}(\forall x_i\neg((A_i(x_i)\wedge B_i(x_i))))\wedge\mathbf{G_r}(\forall x_i\neg((A_i(x_i)\wedge B_i(x_i))))$

The intuitions (that will be formalised in later sections) behind these definitions are the following. COR(A_i, B_i) means that in the current and all future states all things are either both A_i and B_i, or neither A_i nor B_i. IS (A_i, B_i) means that in the current and all future states if anything is A_i then it is also B_i. IS2 (A_i, B_i) means that in the current and all future states if anything is B_i then it is also A_i. DISJOINT (A_i, B_i) means that in the current and all future states there is nothing that is both A_i and B_i.

3.4.2 Extended syntax of the ontology mapping language

A concept A_i in ontology *i* and a concept B_j in ontology *j* can be connected by any of these five mappings, where r is a state, x_i is an instance in ontology *i*, y_j is an instance in ontology *j*. The state operators $\mathbf{N_r}$ and $\mathbf{G_r}$ are defined in section 3.5. and Rel(x_i, y_j) in section 3.6.

Syntax	**Extended syntax**
COR(A_i, B_i)	$\boldsymbol{N_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$ $\boldsymbol{G}_{\mathbf{r}}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j))))$
IS (A_i, B_i)	$\boldsymbol{N_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j)))) \wedge$ $\boldsymbol{G_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j))))$
IS2 (A_i, B_i)	$\boldsymbol{N_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \vee \neg B_j(y_j)))) \wedge$ $\boldsymbol{G_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \vee \neg B_j(y_j))))$
COMPATIBLE (A_i, B_j)	$\boldsymbol{F_r}(\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j)))$
DISJOINT (A_i, B_j)	$\boldsymbol{N_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))) \wedge$ $\boldsymbol{G_r}(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j))))$

The intuitions behind these definitions are analogous to the earlier ones, with a subtle difference that we compare predicates from two ontologies. Also, here we have COMPATIBLE (A_i, B_j) that means that there is a state in the future when something that is A_i (in the language of ontology *i*) is also B_j (in the language of ontology *j*).

3.5 Defining compositional semantics

In order to define the meaning and truth evaluation of the statements above (ontology relationships and ontology mappings) we have to define how these compound statements can be de-composed into elementary ones. Typically, we will want to define how the expression

$V_{i,m}[exp]$

is evaluated, where $V_{i,m}$ is a truth valuation function (where *i* is a context identifier and *m* and state number) that returns either 0 (false) or 1 (true) and where *exp* is one of the statements defined in sections 3.4 and 3.5, i.e. either an ontological relationship or ontology mapping. This truth valuation will also occur in a form with only one index (*m*) and then the valuation is done solely from the point of view of state *m* (and independent of context). Also, it will be done from the point of view of one context (*i*) and independent of state, or from the point of view of a combined context (*i, j*) and independent of state . The truth valuation will also occur in a form with no index at all, and then the valuation is done independently of state and context (that is equivalent to saying it is done individually in all pairs of states and contexts). All these cases will be described in later sections.

The notation below will help us to describe how the original expression *exp* goes through the different stages of decomposition into its elementary constituents.

A formula without quantifiers that could contain conjunctions, disjunctions or implications connecting formulas (we assume that it then is automatically translated into disjunctive normal form) from several different contexts (but all defined in a state *m*) will be denoted as
$<S_I, m>$: *exp*

where S_i is the set of the indexes of all the contexts that appear in S_I (notice that this *exp* is different from the *exp* above, because the earlier one could contain quantifiers but not this one. The *exp* below is yet different, because it is even more restricted.). Another case is when the formula completely belongs to a single context. The combination of state *m* and local context *i* can be written as $<i, m>$ and

$<i, m>$: *exp*

means that all constants and predicates in the expression *exp* belong to $L_{i,m}$ (a propositional language defined in section 3.6) and are interpreted in state *m* and local context *i*. This also covers the case when *i* is a combined context (*j*, *k*) , and then *exp* belongs to $L_{j,m} \times L_{k,m}$.

There is a set of discrete states T, and a relationship $L(s_x, s_y)$ that means that a state s_y succeeds a state s_x (but it does not mean that s_y is the *immediate* successor of s_x). L() is transitive, antisymmetric and irreflexive. An additional constraint is the following one and this means that the states are totally ordered:

$$\forall s_x \in T, s_y \in T(s_x \neq s_y \Rightarrow s_x > s_y \vee s_y > s_x)$$

We then introduce some symbols and they are used as quantifiers over many states, when expressing ontology mappings (i.e. they are a type of practical abbreviations).
If the expression *exp* in $< S_I, z >: exp$ does not have the state parameter *z explicitly* mentioned, i.e. only the context identifier of the concepts is mentioned, then we assume that the concepts are implicitly defined in state z that becomes a parameter. Here, we define the truth value of expressions with state operators, from a point of view of context *i.*

$$\begin{array}{lll}
V_i[\boldsymbol{G}_r(< i, z >: exp)] = 1 & \text{iff } \forall(s' \in S)(L(r, s') \rightarrow V_{i,s'}[exp|_{z=s'}] = 1) & \\
V_i[\mathbf{F}_r(< i, z >: exp)] = 1 & \text{iff } \exists(s' \in S)(L(r-1, s') \wedge V_{i,s'}[exp|_{z=s'}] = 1) & \textbf{(1)} \\
V_i[\mathbf{N}_r(< i, z >: exp)] = 1 & \text{iff } V_{i,r}[exp|_{z=r}] = 1 &
\end{array}$$

We use the notation $exp|_{z=s'}$ to mean that in the expression *exp* we have substituted all occurrences of *z* with *s'*.

When the state operators are applied to an expression that contains predicates from two different contexts *i* and *j* then their meaning is the following. Here, (*i, j*) is a combined context viewpoint from which the truth valuation is done.

$$\begin{array}{lll}
V_{(i,j)}[\boldsymbol{G}_r(< \{i, j\}, z >: exp)] = 1 & \text{iff } \forall(s' \in S)(L(r, s') \rightarrow V_{(i,j),s'}[exp|_{z=s'}] = 1) & \\
V_{(i,j)}[\mathbf{F}_r(< \{i, j\}, z >: exp)] = 1 & \text{iff } \exists(s' \in S)(L(r-1, s') \wedge V_{(i,j),s'}[exp|_{z=s'}] = 1) & \textbf{(2)} \\
V_{(i,j)}[\mathbf{N}_r(< \{i, j\}, z >: exp)] = 1 & \text{iff } V_{(i,j),r}[exp|_{z=r}] = 1 &
\end{array}$$

Why have we defined the state operators like this? Our inspiration is (Gamut 1991, page 33) and our definitions of $\mathbf{G_r}$ and $\mathbf{F_r}$ are similar to theirs, except 1) that they do not include the notion of context in the normal definition of these operators, and 2) instead of including the state variable *r* inside the operators they make valuation of the operators state-dependent, e.g. the valuation of **G** would then say that it is true in all states after the one from which the valuation is done. The reason we chose another definition is because it corresponds better to our application where ontology relationships and mappings are created in certain states and then remain, unless they are deleted. So the fact that that a certain ontology relationship or mapping is created in a certain state is independent of state.

Here follows a definition that takes care of the case where a quantifier precedes the statement. The reason we need these definitions is for defining ontology mappings – they always contain one of these operators. Four different cases are presented.

$V_m[\forall v_i < S_I, m >: exp] = 1$ assuming $i \in S_I$ iff **(3)**

For every $d \in D_{i,m}$, if $I_{i,m}(v_i) = d$ then $V_m[< S_I, m >: exp(v_i)] = 1$

Otherwise, $V_m[\forall v_i < S_I, m >: exp] = 0$

$V_m[\forall v_i v_j < S_I, m >: exp] = 1$ assuming $i, j \in S_I \;\wedge\; i \neq j$ iff **(4)**

For every pair $d \in D_{i,m}$, $d' \in D_{j,m}$ if $I_{i,m}(v_i) = d$ and $I_{j,m}(v_j) = d'$ then

$V_m[< S_I, m >: exp(v_i, v_j)] = 1$

Otherwise, $V_m[\forall v_i v_j < m, S_I >: exp] = 0$

$V_m[\exists v_i < S_I, m >: exp] = 1$ assuming $i \in S_I$ iff **(5)**

There is at least one $d \in D_{i,m}$ and $I_{i,m}(v_i) = d$ such that $V_m[< S_I, m >: exp(v_i)] = 1$

Otherwise, $V_m[\exists v_i < S_I, m >: exp] = 0$

$V_m[\exists v_i v_j < S_I, m >: exp] = 1$ assuming $i, j \in S_I \;\wedge\; i \neq j$ iff **(6)**

There is at least a pair $d \in D_{i,m}$, $d' \in D_{j,m}$ and $I_{i,m}(v_i) = d$ and $I_{j,m}(v_j) = d'$ such that

$V_m[< S_I, m >: exp(v_i, v_j)] = 1$

Otherwise, $V_m[\exists v_i v_j < S_I, m >: exp] = 0$

We shall now present a recursive definition of the truth value of a compound expression that is a conjunction or disjunction of formulas from several different contexts. At this stage all the state operators, implications and quantifiers have been removed from the expression that we evaluate (using the rules above). Here, the truth valuation is made from a state *m* and then it evaluates all sub-expressions inside their individual contexts.

$$V_m[<S_I, m>: exp] = Max(V_{i,m}[<i,m>: exp], V_m[<S'_I, m>: exp]) \quad \textbf{(7)}$$

where $i \in S_I$ and $S'_I = S_I / \{i\}$

So it means that we use recursion to evaluate the sub-expression that belongs to a context *i* and then evaluate the remaining part of the expression (i.e. formulas that *do not* belong to context *i*)

Another special case is the equivalence (assuming $i \neq j$):

$$V_m[<i,m>: exp_1 \leftrightarrow <j,m>: exp_2] = 1 \text{ iff} \quad \textbf{(8)}$$

$$V_{i,m}[<i,m>: exp_1] = V_{j,m}[<j,m>: exp_2]$$

The final special case is the implication (assuming $i \neq j$):

$$V_{(i,j),m}[<(i,j),m>: (exp_1 \rightarrow exp_2)] = 1 \text{ iff} \quad \textbf{(9)}$$

$$V_{(i,j),m}[<(i,j),m>: (\neg exp_1 \vee exp_2)] = 1$$

If we now behold the statements from section 3.4 and procedures from section 3.5, then we see that if we apply the decomposition rules in this section, all state operators, quantifiers and implications will be removed. We will now summarize the decomposition procedure that takes statements from section 3.4.1 (ontology relationships) and 3.4.2 (ontology mappings) as the starting point. Procedures (1) and (2) remove all state operators, procedures (3), (4), (5) and (6) remove all quantifiers, and finally (8) and (9) remove equivalence and implication symbols.

3.6 Definition of the formal semantics (model theory)

By using the rules from section 3.5 we have now decomposed the truth valuation of compound statements from section 3.4 into truth valuation of expressions *vexp* (these expressions have a simple structure that is described below) with the following limited language:

$$vexp = <m, S_I>: exp$$

where

$$vexp = vexp \wedge vexp \,|\, vexp \vee vexp \,|\, \neg vexp \,|\, < i,m >: A_i(x_i) \,|\, \text{<(i,j),m>:Rel}\left(\text{b}_\text{i},\text{d}_\text{j}\right)$$

We will therefore present the definitions that are needed to evaluate the meaning and truth value of all possible expressions *vexp*.

$L_{i,m}$ is the language used in state *m* and context *i* and it is specified by a set of unary predicated P_i^1, P_i^2 ... P_i^n and a set of constants $a^1, a^2..., a^m$

$P(a)$ and $\neg P(a)$ are well-formed formulas in F.

If F_1 and F_2 are well-formed formulas then $F_1 \wedge F_2$ and $F_1 \vee F_2$ are also well-formed formulas F.

A model **M** for a set of languages $L_{i,m}$ (m=0, 1, … and this corresponds to the states s_0, s_1, …, and i=0, 1, … and this corresponds to local contexts c_0, c_1, …) consists of a set of domains $D_{i,m}$ (that are non-empty sets, and m=0, 1, … and i=0, 1, …) and a set of interpretation functions $I_{i,m}$ (m=0, 1, … and i=0, 1, …) which are defined on the set of instances and predicate names in the vocabulary of $L_{i,m}$ and adhere to the following rules below. We first define how to evaluate variables.

g is an assignment that assigns an individual to each variable. It is defined in this way:

$[\![t]\!]_{M,g}$ = $I_{i,m}(t)$ if t is a constant in $L_{i,m}$.
g(t) if t is a variable

If x is an constant in $L_{i,m}$ then $I_{i,m}(x_i) \in D_{i,m}$

If P is an n-ary predicate name in $L_{i,m}$, then $I_{i,m}(P_i) \subseteq D_{i,m}^n$.

The truth value of any possible formula is defined in this way (we implicitly assume that the truth valuation function $V_{i,m}$ uses the model **M**):

If $P(t)$ is an atomic sentence in $L_{i,m}$ (i.e. the language of local context c_i in state s_m), then

$V_{i,m}[P(t)] = 1$ if and only if $[\![t]\!]_{M,g} \in D_{i,m}$ and $\langle [\![t]\!]_{M,g} \rangle \in I_{i,m}(P)$ and

$V_{i,m}[P(t)] = 0$ if and only if $[\![t]\!]_{M,g} \in D_{i,m}$ and $\langle [\![t]\!]_{M,g} \rangle \notin I_{i,m}(P)$.

$V_{i,m}[\neg P(t)] = 1$ if and only if $V_{i,m}[P(t)] = 0$

$V_{i,m}[\neg P(t)] = 0$ if and only if $V_{i,m}[P(t)] = 1$.

Given a well-formed formula (as defined above) having the form $P \wedge Q$ in the state s_m and local context c_i, its truth value (using the assignment g and model **M**) is

$V_{i,m}[P \wedge Q] = 1$ iff $V_{i,m}[P] = 1$ and $V_{i,m}[Q] = 1$

$V_{i,m}[P \wedge Q] = 0$ otherwise

Given a valid formula (as defined above) having the form $P \vee Q$ in state s_m and local context c_i, its truth value value (using the assignment g and model **M**) is

$V_{i,m}[P \vee Q] = 1$ iff $V_{i,m}[P] = 1$ or $V_{i,m}[Q] = 1$

$V_{i,m}[P \vee Q] = 0$ otherwise

If $P(t)$ belongs to $L_{i,m}$ then

the truth value of $V_{j,m}[P(t)]$ is undefined iff $j \neq i$.

We then introduce the Rel() operator where

$\mathrm{I}_{(i,j),m}(\mathrm{Rel}(x_\mathrm{i}, y_\mathrm{j})) \in D_{i,m} \times D_{j,m}$

and its truth valuation (from a combined context (i,j))

$V_{(i,j),m}[\mathrm{Rel}(x_\mathrm{i}, y_\mathrm{j})] = 1$

If $[\![x_i]\!]_{M,g} \in D_{i,m} \wedge [\![x_i]\!]_{M,g} = e_1$ and

$[\![y_j]\!]_{M,g} \in D_{j,m} \wedge [\![y_j]\!]_{M,g} = e_2$ and $e_1 = e_2$

Otherwise, $V_{(i,j),m}[\mathrm{Rel}(x_\mathrm{i}, y_\mathrm{j})] = 0$

This definition makes sense if we consider that the domains $D_{1,s}$, $D_{2,s}$, …, actually overlap sometimes. We assume that there is a domain D that contains all the local domains $D_{i,m}$ (*i* traverses all local contexts, and *m* traverses all existing states).

Also, because the domains $D_{1,s}$, $D_{2,s}$, …, actually can overlap, we can view them as subsets of a domain D_s – i.e. what is true in a given state *s* independently of what the local contexts can see is true. However, we never use the domain D_s to create a centralised knowledge representation as such (due to epistemological assumptions discussed in section 3.2).

This special case is also true:

$$\mathrm{Rel}\left(\mathrm{b}_i, \mathrm{d}_i\right) = true \text{ if } \mathrm{b{=}d}$$

Because every instance corresponds to itself.

As the next step we have to define the truth value of compound expressions that contain formulas from different ontologies.

We now add the following constraints to our logical formalism, where $\oplus$ means exclusive or.

$$\forall m(m \geq 0 \rightarrow (\mathrm{I}_{i,m+1}(<m+1, i>: b_i) = \mathrm{I}_{i,m}(<m, i>: b_i)) \oplus b_i \notin L_{i,m+1} \oplus b_i \notin L_{i,m})$$

$$\forall m(m > 0 \rightarrow (\mathrm{I}_{i,m+1}(<m+1, i>: P_i) = \mathrm{I}_{i,m}(<m, i>: P_i)) \oplus P_i \notin L_{m+1,i} \oplus P_i \notin L_{m,i})$$

The meaning of the two statements is that a constant that has a certain name retains the same meaning in the next stage unless it is deleted in the latter states or did not exist in the former state and that a predicate (that corresponds to a concept) retains both its existence and meaning in the next stage, unless it is deleted in the latter state or did not exist in the former state.

The mappings as such have a logical objective existence, although they map between different local (subjective) models. It would have been possible to investigate if the mappings as well should only have local existence, e.g. only exist within the local contexts (see for example [9]), but this would have made the formalism more complex. Instead, we want to focus on the dynamic aspects of the system and it is therefore more important that a certain local context is *responsible* for having proposed that a certain mapping is created (and this is not visible in the model semantics as such). The contexts therefore exhibit agent-like behaviour, because they have a form of autonomy, but this autonomy to change knowledge is not visible in the model semantics (but the formalism in chapter 5 will illustrate it).

3.7 The relationship between our ontology language and OWL

Because OWL is an accepted standard (despite not the only one) for expressing ontologies on the web we mention how a small subset of OWL Axioms could be mapped to our logic and vice versa.

OWL Axiom	Our language	**OWL Axiom**	Our language
$C \doteq D$	COR(C_j, D_j)	$C \sqsupseteq D$	IS2 (C_j, D_j)
$C \sqsubseteq D$	IS (C_j, D_j)	$C \sqsubseteq \neg D$	DISJOINT (C_j, D_j)

The first of these relationships is equivalence between two concepts, the second means "is subsumed by", the next "subsumes" and the last one expresses disjointness.
If we assume that a certain relationship was created in a certain state, then we can translate axioms written in OWL to our own notation, using the table above. C and D denote *concept names* in OWL. So if OWL axioms that form an ontology are imported in state *r* then that ontology is re-created in our logical language in state *r*.

3.8 The relationship between our ontology mapping language and C-OWL

Let us now look at the ontology mappings, and compare them to the mapping language C-OWL. These are five possible ontology mappings in our logic between a concept A in ontology *i* and a concept B in ontology *j*:

- COR(A_i, B_j)
- IS (A_i, B_j)
- IS2 (A_i, B_j)
- DISJOINT (A_i, B_j)
- COMPATIBLE (A_i, B_j)

For example COR() describes that in all future states two concepts from two different ontologies will have the same meaning, whereas IS() expresses subsumption across ontologies (that will persist in future states). COMPATIBLE() is a logically “weak” relation between two concepts.
Our mappings approximately correspond to (but C-OWL does not have a temporal notion of state, and our logic does not have directionality) and could be created by importing the following 5 C-OWL bridge rules (Bouquet et al 2003):

$$i{:}A \xrightarrow{\equiv} j{:}B,\quad i{:}A \xrightarrow{\sqsubseteq} j{:}B,\quad i{:}A \xrightarrow{\sqsupseteq} j{:}B,\quad i{:}A \xrightarrow{\perp} j{:}B,\quad i{:}A \xrightarrow{*} j{:}B$$

The rightmost C-OWL bridge rule here corresponds to our relationship COMPATIBLE().

Chapter 4 A computational framework for combining logical elements and reasoning about consistency and redundancy

4.1 Introduction

In Chapter 3 we described a formal theory that defines the formal meaning of ontology mappings and ontologies themselves. However, that theory becomes useful if we can do computations with it and reason about ontologies and mappings. We firstly describe the motivation behind these computations and we then show how to map the semantic meaning of ontologies and mappings to computational elements that can be processed by algorithms with good computational complexity (that will be quantified). When we write "mapping/relationship" then we refer to the computational element to which both ontology mappings and relationships within ontologies are mapped. "Ontology mapping" is a phrase used in the ontology community that we will use here as well, despite the fact that ontology mappings are not mappings in the mathematical sense, but rather relations.

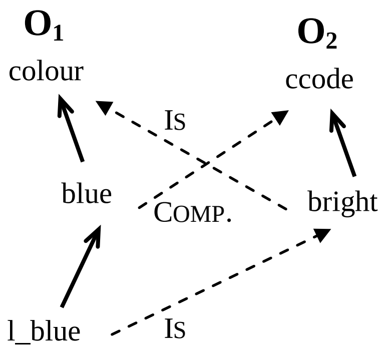


Figure 6. Two example ontologies (connected by mappings) we will reason about.

4.2 Motivation

In order to be able to evolve a set of ontologies (currently, one ontology at a time) and mappings and at the same time maintain consistency between them where it is needed, we formalize the ontologies and the mappings between them. We mention a very simple example problem that we will try to solve using the proposed formalization and proof methods.

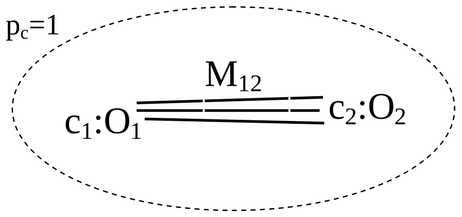


Figure 7. An illustration of the sphere of consistency encapsulating both ontologies (O_1 and O_2) and the set of mappings (M_{12}) connecting them.

Here follows a simple example that is illustrated in Figure 6. We have two ontologies O_1 and O_2, and the first ontology

contains the concepts of the set {colour, blue, l_blue} and the second one the concepts of the set {ccode, bright}. We also have the following subsumption hierarchies within the ontologies: Is($blue_1$, $colour_1$), Is(l_blue_1, $blue_1$), Is($bright_2$, $ccode_2$). Then we define these mappings between the two ontologies: COMPATIBLE($blue_1$, $ccode_2$), Is(l_blue_1, $bright_2$), Is($bright_2$, $colour_1$). We want to investigate if changes in such system alter consistency. When we say that "the system is consistent" in this situation we actually imply that both ontologies and their mappings taken together are consistent. We will represent this by defining a so called sphere of consistency that includes both ontologies and the set of mappings between them. It is illustrated in Figure 7 and we assume in this example that we want to maintain perfect consistency – as we will see, that is formalised by setting a consistency parameter p_c to 1. Now we would like to know if this system, given these circumstances, is consistent with the mappings COMPATIBLE($blue_1$, $bright_2$) or DISJOINT($blue_1$, ccode). We therefore have to investigate two separate cases, and the answer can be either yes or no, in each case.
Real scenarios will assume more and larger ontologies and more mappings.

4.3 Spheres of consistency

4.3.1 Defining Spheres of consistency

In a large knowledge infrastructure that has many ontologies it is sometimes infeasible for the reasoning to spread across the whole infrastructure, so it sometimes should be bound to knowledge regions that consist of ontologies and mappings between them. Additionally, as we have motivated in (Zurawski et al. 2008), sometimes, we could be interested in only be looking for contradictions that require less computations to find and that are more obvious – this corresponds to contradictions that have smaller proofs. Further motivation of proof-bounded consistency is to be found at the end of §4.3.2.

Therefore, we now define spheres of consistency, in the following way:

Given a set of contexts {c_1, c_2, c_3, … }
where every context c_i has an ontology O_i
and given a set of mapping sets {m_{12}, m_{13}, m_{23}, … }
where every m_{ij} is the set of all mappings connecting contexts *i* and *j* where *i*<*j*,
a sphere of consistency is defined as

$$Cons(\{c_i, c_j, ...\}, \{m_{ij}, m_{ik}, ...\}, p_c, p_r) \tag{1}$$

where $\{c_i\ldots\}$ and $\{m_{ij}\ldots\}$ are defined as above and p_c is a continuous consistency parameter that can vary between

$p_c=1$ which means full consistency, and
$p_c=2$ which means that inconsistencies of all depths are fully allowed

When $1<p_c<2$ then the sphere of consistency defines a proof-bounded consistency, where there is no proof of contradiction where the proof search tree has a smaller depth than d (that must be an integer), and p_c and d are related through the following formula:

$$p_c = 2 - \frac{d-2}{tot_s - 1} \tag{2}$$

where tot_s is the total amount of relationships in all the ontologies and mapping sets that are included in the sphere of consistency. In an analogous way we define $1 \le p_r \le 2$ that measures the amount of bounded redundancy within a sphere of consistency where there is no proof of redundancy where the proof search tree has a smaller depth than d, and p_r and d are related through this formula:

$$p_r = 2 - \frac{d-2}{tot_s - 1} \tag{3}$$

4.3.2 Explaining and motivating the definition of spheres of consistency

The reasoning layer detects if a proposal would cause a contradiction or redundancy within some specified spheres of consistency. If consistency has to be maintained within a reasoning space that has ontologies and/or mappings (that in total contain tot_s relationships) then in the worst case the depth of the reasoning proof search tree will be $d=tot_s+1$ because if there is no contradiction smaller than $d=tot_s+1$, complete consistency can be guaranteed. The reason for this is that the search space is limited and this particular reasoning algorithm uses a search method that systematically will explore this search space, without unnecessary loops (because both Algorithm A and B have loop-prevention mechanisms, as described in §4.12.3 and §4.12.4) . If we use d=tot$_s$+1 in formula (2) then p_c=1, i.e. full consistency, but d=2 will give the result p_c=2, i.e. all inconsistencies are possible. The smallest contradiction that our reasoning recognizes has size 2 [e.g. Figure 8 shows a contradiction of size 4] and if contradictions of that size are accepted then all operations that would introduce contradictions are accepted.

The reason why we have chosen to define consistency in terms of the depth of the proof search tree is that we can make sure that the system holds this property *incrementally* when it moves to the next state. Before the whole system S starts to evolve the user has to *define* all spheres of consistency, their regions (i.e. the sets in formula (1)) and their individual degrees p_c and p_r. Then there must be some expectation of the size the system will reach, and using that expectation (as tot$_s$) the reasoning layer will then calculate d in every sphere using formula (2) but solving for d. Then a proposed change that would introduce a contradiction of length d or smaller would be discovered by the reasoning mechanism that always investigates all possible contradictions (that the proposed change would create) starting with the *small ones*.

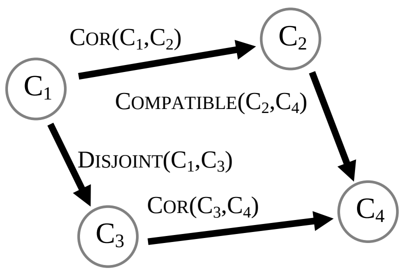


Figure 8. An example of contradiction between concepts C_1 and C_4.

Finally, one should note that an inconsistency that has a proof tree that is more shallow, is a more serious one because it is very direct (e.g. when the size is 2), whereas if it requires a more extensive proof then there are more choices for how to resolve it and a smaller proportion of the relationships have to be removed. One could also claim that in organizational policies the “obvious” contradictions first have to removed, whereas the more subtle ones are discovered later. Also, in the agent-related theory of “bounded-rationality” agents are required only to be able to achieve tasks that require a limited amount of reasoning – and that is true for smaller contradictions.

4.4 Providing Functionality of the Reasoning Layer

The reasoning methods described here provide the functionality of the reasoning layer. To summarize, this is the vocabulary of the reasoning layer:

Expression	**Meaning**
F: C_CONTRA(*sp*, *ont_op*)	returns true if *ont_op* would have introduced a contradiction in sphere *sp* of degree d_c that is higher than the defined degree p_c in that sphere.
F: IS_INFERABLE(*sp*, *ont_op*)	returns true if *ont_op* would have introduced a redundancy in sphere *sp* of degree d_r that is higher than the defined degree p_r in that sphere.
F: IS_NEW(*sp*, *ont_op*)	returns true if neither F: C_CONTRA(*sp*, *ont_op*) or F: IS_INFERABLE(*sp*, *ont_op*) are true.

We will describe the algorithms that provide this functionality.

The current list of operations that evolve ontologies or mappings between them is the following:

ont_op = add_mapping(m, c_j, c_k) │ add_ontorel(m, c_j, d_j) │ delete_mapping(m, c_j, c_k) │ delete_ontorel(m, c_j, d_j) │ ε

4.5 Formal computational notation

4.5.1 Definition of the computational notation and its connection to the formal semantics

We will firstly define the formal computational notation and then we show how algorithms will use it for reasoning, such as the one mentioned. This means that we will have a procedure for combining any two arbitrary relationships/mappings using (from our language) using composition into a new relationship/mapping (that also can be another expression or nothing), so we define a partial composition operator (illustrated visually in the figure above). If the process of reasoning and deciding if a new relationships/mapping is consistent with a set of existing ones or creates redundancy with respect to them only requires this sub-procedure that can combine any two adjacent[3] relationships/mappings into a new one, then there is a procedure for whole task (of verifying consistency) itself. The fact that a new proposed relationship/mapping is consistent with respect to existing ones, means that it does not generate a contradiction when combined with these existing entities.

[3]This means that they have at least one variable in common.

Call every relationship/mapping between two concepts within one ontology or mapping between two concepts in different ontologies, m_i, where i is its unique identifier.

A relationship m_i that holds between the concept C_1 in ontology j and the concept C_2 in the ontology k, can always be expressed using the following general form (consisting of two different types of definitions):

$$m_i(C_{1j}, C_{2k}) = op(\alpha(f(C_{1j}, C_{2k}))) \text{ or}$$

$$m_i(C_{1j}, C_{2k}) = op(\alpha(f(C_{1j}, C_{2k}))) \wedge op'(\alpha'(f'(C_{1j}, C_{2k})))$$

$op \in \{N_a, F_b, G_c\},\ a, b, c \in S$ (the set of states)

$\alpha(f) \in \{\alpha_1(f), \alpha_2(f)\},$ and $R = \mathrm{Re}\,l(x_i, y_j)$

if $i \neq j$ $\quad \{\alpha_1(f), \alpha_2(f)\} = \{\forall x_i y_j (R \to (f)), \exists x_i y_j (R \wedge (f))\}$

if i=j $\quad \{\alpha_1(f), \alpha_2(f)\} = \{\forall x_i (f), \exists x_i (f)\}$

$$f(e_1, e_2) \subseteq \{\langle e_1 \wedge e_2 \rangle, \langle e_1 \wedge \neg e_2 \rangle, \langle \neg e_1 \wedge e_2 \rangle, \langle \neg e_1 \wedge \neg e_2 \rangle\}$$

Rel(x_i,y_i) was defined in section 3.6. Here, *op* is a state operator, $\alpha(\lambda)$ one of two quantifier operators and $f(e_1, e_2)$ a Boolean function with two variables (this particular notation will be described later in this section and translated to standard first order logic).
These expressions only restate the formal expressions from §3.4 by subdividing the formal expressions in three parts and describing every part by a more minimal description, that can be traced back to the formal expression if needed (by using the definitions above) – an example of this is to be found in section 4.5.2.
So the first part consists of a state operator symbol that is mapped to state operators defined before. The second part consists of a quantifier operator and it refers to one of two expressions that begin with quantifiers (in the case of mappings between two concepts from different ontologies) or two other but similar expressions that begin with quantifiers (in the case of relationships between two concepts in one ontology). This means that we map two kinds of formal expressions to the same computational notation. If we want to map the computational notation of these quantifiers (i.e. α_1 or α_2) back to the formal notation we

have to look at the two variables (C_{1j} and C_{2k}). If *j=k* then α_1 and α_2 are mapped to the second formal expression, whereas if $j \neq k$ then they are mapped to the first formal expression. This is natural if one considers the definition of R in section 3.6 and that

$$\forall x_i y_j (R \rightarrow (\lambda)) \wedge R = \mathrm{Rel}(x_i, y_j) \Leftrightarrow \forall x_i(\lambda)$$
$$\exists x_i y_j (R \wedge (\lambda)) \wedge R = \mathrm{Rel}(x_i, y_j) \Leftrightarrow \exists x_i(\lambda)$$

The third part of the expression above is a Boolean function in DNF-form that has two variables and can contain four elements (none, some or all of them). These elements are actually connected using disjunction, so e.g. $f(e_1,e_2) = \{\langle e_1 \wedge e_2\rangle, \langle e_1 \wedge \neg e_2\rangle\}$ using this notation corresponds to $f(e_1,e_2) = (e_1 \wedge e_2) \vee (e_1 \wedge \neg e_2)$ using conventional notation. Two other examples are $f(e_1,e_2) = \{\} = false$ and $f(e_1,e_2) = \{\langle e_1 \wedge e_2\rangle, \langle e_1 \wedge \neg e_2\rangle, \langle \neg e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle\} = tru$. So these last two examples cover the extreme cases.
A technical detail is that we every time we encounter the second type of relationship/mapping (i.e. $m_i(C_{1j},C_{2k}) = o(p\alpha(f(C_{1j},C_{2k}))) \wedge o'(p\alpha'(f'(C_{1j},C_{2k})))$, we split it into two separate mappings that will be processed separately.

4.5.2 Restating the originally proposed types of mappings and relationships

The computational notation above allows the creation of a huge variety of mappings (between ontologies) or relationships (within ontologies). Currently we will however stick to the five originally proposed mapping types and four relationship types and restate them in this new concise language. If we use the definitions from §4.5 then we will see that both the four ontology relationships from §3.4.1 and five ontology mapping types from §3.4.2 can be mapped to the following shared computational notation. In the following notation, *r* is a state, e$_1$ is the name of the logical predicate that corresponds to the concept C$_{1j}$ and e$_2$ is the name of the logical predicate that corresponds to the concept C$_{2k}$. I.e. e$_1$=C$_{1j}$(x) and e$_2$=C$_{2j}$(y).

1. CORRESPONDENCE [COR.] (C$_{1j}$, C$_{2k}$)

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle\}$$

2. IS (C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle \neg e_1 \wedge e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle \neg e_1 \wedge e_2\rangle\}$$

3. IS2 (C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle e_1 \wedge \neg e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle e_1 \wedge \neg e_2\rangle\}$$

4. DISJOINT (C_{1j}, C_{2k})

$$N_r\alpha_1\{\langle \neg e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle e_1 \wedge \neg e_2\rangle\} \wedge G_r\alpha_1\{\langle \neg e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle e_1 \wedge \neg e_2\rangle\}$$

5. COMPATIBLE (C_{1j}, C_{2k})

$$F_r\alpha_2\{\langle e_1 \wedge e_2\rangle\}$$

As we will see in the applications of this formalism, *r* is instantiated to the value of the state when these relationships/mappings are created.
We now show an example where we take a simple mapping between two concepts, unfold the computational notation and actually write explicitly what it means formally. This will not be repeated, because the full-length formalization is very long and impractical to use explicitly, when we want to combine mappings.

If we return to the example in §4.2, then we know that Is(l_blue$_1$, bright$_2$) and using the computational notation it can be briefly described as below. In this example we will assume that the mapping was created in state 7 (i.e. *r*=7). We will remind the reader that the index *i* refers to things that belong to ontology O_i whereas index j refers to ontology O_j and that s´ is a state variable that is used to refer to a range of states (possibly infinitely many). The variable *z* is an implicit variable that decides in which state the predicates inside the valuation functions are defined and evaluated.

Is(l_blue$_1$, bright$_2$) $\Leftrightarrow$ *(definition for Is() from above)*

$$N_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle \neg e_1 \wedge e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle, \langle \neg e_1 \wedge \neg e_2\rangle, \langle \neg e_1 \wedge e_2\rangle\} \Leftrightarrow$$

$$N_7\alpha_1\{\langle L\_blue_1(x_1) \wedge Bright_2(y_2)\rangle, \langle \neg L\_blue_1(x_1) \wedge \neg Bright_2(y_2)\rangle, \langle \neg L\_blue_1(x_1) \wedge Bright_2(y_2)\rangle\} \wedge$$
$$G_7\alpha_1\{\langle L\_blue_1(x_1) \wedge Bright_2(y_2)\rangle, \langle \neg L\_blue_1(x_1) \wedge \neg Bright_2(y_2)\rangle, \langle \neg L\_blue_1(x_1) \wedge Bright_2(y_2)\rangle\} \Leftrightarrow$$

$$V_{(1,2),7}[\alpha_1((L\_blue_1(x_1) \wedge Bright_2(y_2)) \vee (\neg L\_blue_1(x_1) \wedge \neg Bright_2(y_2)) \vee$$
$$(\neg L\_blue_1(x_1) \wedge Bright_2(y_2)))|_{z=7}] = 1 \wedge$$
$$\forall(s' \in S)(L(7,s') \rightarrow V_{(1,2),s'}[\alpha_1((L\_blue_1(x_1) \wedge Bright_2(y_2)) \vee$$
$$(\neg L\_blue_1(x_1) \wedge \neg Bright_2(y_2)) \vee (\neg L\_blue_1(x_1) \wedge Bright_2(y_2)))|_{z=s'}] = 1) \Leftrightarrow$$

$$\forall(s' \in S)(L(6,s') \rightarrow V_{(1,2),s'}[\forall x_1 y_2(\mathrm{Rel}(x_1,y_2) \rightarrow ((L\_blue_1(x_1) \wedge Bright_2(y_2)) \vee$$
$$(\neg L\_blue_1(x_1) \wedge \neg Bright_2(y_2)) \vee (\neg L\_blue_1(x_1) \wedge Bright_2(y_2)))))|_{z=s'}] = 1)$$

4.6 Proof Methodology

We will described the proof task that has to be carried out in order to deliver the functionality (described in §4.4) that is required by the statements F: C_CONTRA(*sp*, *ont_op*), F: IS_INFERABLE(*sp*, *ont_op*) and F: IS_NEW(*sp*, *ont_op*), where *sp* is the sphere of consistency where the reasoning is done. This means that all the elements (both ontologies and ontology mappings) from that sphere of consistency are included in the reasoning process in their computational form.

4.6.1 The general proof tasks

There is a sphere of consistency *sp* (these were defined in §4.3) and it contains a set of ontologies **O**={O_1, O_2, O_3, …} and a set of concepts **C**={C_{11}, C_{12}, C_{22} …, C_{nm}}. Every concept C_{ax} belongs to a single ontology O_x.

Then, every possible case that needs to be proved has the following form, where $m_1 \dots m_n$ is the set of existing relationships/mappings (in computational form) and m_{n+1} is the new proposed mapping.

M is the following set:

$m_1(C_{ax}, C_{by})$ and $C_{ax} \in O_x$, $C_{by} \in O_y$

$m_2(C_{ax}, C_{by})$ and $C_{ax} \in O_x$, $C_{by} \in O_y$

.

.

$m_n(C_{ax}, C_{by})$ and $C_{ax} \in O_x$, $C_{by} \in O_y$

M' is then the following element:

$$m_{n+1}(C_{ax}, C_{by}) \text{ and } C_{ax} \in O_x,\ C_{by} \in O_y$$

Proof task 1. Calculating if a proposed change introduces a contradiction.

The proof task of F: C_CONTRA(*sp*, *ont_op*) is then to test mechanically if performing the ontology operation *ont_op* on the set **M** in the sphere of consistency *sp* will result in a contradiction of degree d_c that is higher than the defined degree p_c in that sphere. The answer will be true or false. As a special case, if p_c=1, then the algorithm will answer if *any* contradiction would be generated by the operation *ont_op.*

Proof task 2. Calculating if a proposed change introduces a redundancy.

The proof task of F: IS_INFERABLE(*sp, ont_op*) is then to test mechanically if performing the ontology operation *ont_op* on the set **M** in the sphere of consistency *sp* will result in a redundancy of degree d_r that is higher than the defined degree p_r in that sphere. The answer will be true or false. As a special case, if p_r=1, then the algorithm will answer if *any* redundancy would be generated by the operation *ont_op.* A redundancy is introducing a piece of knowledge (here: relationships/mapping) M' that could already by inferred from **M** by applying the rules of inference described in this chapter.

Proof task 3. Calculating if a proposed change introduces a change that neither generates a contradiction or redundancy.

The proof task of F: IS_NEW(*sp, ont_op*) is then to test mechanically if performing the ontology operation *ont_op* on the set **M** in the sphere of consistency *sp* will result in that a change that neither generates a contradiction or redundancy (of degree d_c that is higher than the defined degree p_c in that sphere). The answer will be true or false.

Given a sphere of consistency *sp* and ontology operation *ont_op*, then if F: C_CONTRA(*sp*, *ont_op*) returns false and F: IS_INFERABLE(*sp*, *ont_op*) returns false, then F: IS_NEW(*sp*, *ont_op*) will return true.

4.6.2 A proof task example

In the aforementioned example (from §4.2) we have to investigate whether these two sets of ontology relationships (belonging to ontology O_1 and O_2 respectively)

O_1

Is($blue_1$, $colour_1$)

Is(l_blue_1, $blue_1$)

O_2

Is($bright_2$, $ccode_2$)

and this set of mappings

COMPATIBLE($blue_1$, $ccode_2$)

Is(l_blue_1, $bright_2$)

Is($bright_2$, $colour_1$)

is consistent with adding

COMPATIBLE ($blue_1$, $bright_2$)

I.e. that F: C_CONTRA (*sp*, add_mapping(COMPATIBLE ($blue_1$, $bright_2$), c_1, c_2)) is false, where sp is the set with all the three ontology relationships and three ontology mappings above. c_1 is then the context proposing to add the mapping from its ontology to that of context c_2.

And then we have to test if the same set of six mappings is consistent with

DISJOINT($blue_1$, $ccode_2$)

I.e. that F: C_CONTRA (*sp,* add_mapping(DISJOINT(blue$_1$, ccode$_2$), c$_1$, c$_2$)) is false, where sp is the set with all the three ontology relationships and three ontology mappings above.
Moreover, let us assume that the set of mappings was created during the states 1 to 6, whereas the proposed mapping (i.e. Compatible in the first test, and Disjoint in the second test) would belong to state 7.
In order to be able to reason about what happens when two or more mappings are combined (that is a step in the reasoning process), we have to take a look at rewrite rules. The rewrite rules will apply to the three parts of every relationship/mapping (as described in §4.5). It is therefore natural that there will be three kinds of mechanical processes, because they process one part each, and these processes (e.g. rewrite rules) will be described in sections 4.7, 4.8 and 4.9 Section 4.10 describes how to combine their results.

4.7 Rewrite Rules for state operators

We will now describe the steps of the mechanical procedure that we have outlined in §4.5.1. where two relations are composed. The first part of that procedure is to compose states. Therefore, the first set of rewrite rules focus on the state operators. A comment about notation: In this thesis we use the arrow $\Rightarrow$ for two different purposes. Firstly, it is used when describing mechanical rewrite rules (as we do below). Secondly, it is used for showing derivation steps in proofs using first order logic. The context should make it clear which meaning it has.

We will now use the following notation:

$\Phi\Lambda \Rightarrow \Theta$, where $\Phi,\Lambda \in \{N_a, G_b, F_c, \varepsilon\},\ a,b,c \in S$,

This means that the two symbols on the left of the $\Rightarrow$ are rewritten to the symbol on the right side, and this corresponds to composing these symbols (we see an illustration of this at the end of §4.10, and if e.g. the first symbol operates on the arguments e$_1$ and e$_2$, the second on the arguments e$_2$ and e$_3$, then the resulting symbol operates on e$_1$ and e$_3$).
The *N, G* and *F* symbols have been defined before. The ε operator means that it is possible (but not necessary) that there are one or more states now or in the future where the expression is true, but that none of the operators N_a, G_b, F can be inferred to hold there.

Now we can easily describe the how the operators are combined in the nine possible standard cases:

$N_a N_b \Rightarrow N_a$ if $a = b$ or ε if $a \neq b$
$N_a G_b \Rightarrow \varepsilon$ if $a \leq b$ or N_a if $a > b$
$N_a F_b \Rightarrow \varepsilon$

$G_a N_b \Rightarrow \varepsilon$ if $b \leq a$ or N_b if $b > a$
$G_a G_b \Rightarrow G_{\max(a,b)}$
$G_a F_b \Rightarrow F_b$ if $b \geq a$ or ε if $b < a$

$F_a N_b \Rightarrow \varepsilon$
$F_a G_b \Rightarrow F_a$ if $a \geq b$ or ε if $a < b$
$F_a F_b \Rightarrow \varepsilon$

The non-standard cases are these:

$N_a \varepsilon \Rightarrow \varepsilon,\ G_a \varepsilon \Rightarrow \varepsilon,\ F_a \varepsilon \Rightarrow \varepsilon,$
$\varepsilon N_a \Rightarrow \varepsilon,\ \varepsilon G_a \Rightarrow \varepsilon,\ \varepsilon F_a \Rightarrow \varepsilon,\ \varepsilon\varepsilon \Rightarrow \varepsilon$

Note that the order in which the operators are combined does not matter, i.e. if $\Phi\Lambda \Rightarrow \Theta$ and $\Phi, \Lambda \in \{N_a, G_b, F_c, \varepsilon\}$ then $\Lambda\Phi \Rightarrow \Theta$. It seems that the ε operator acts like a "zero" element because everything it operates on only results in a ε.

We should note that when two operators are combined (in an arbitrary order) they result in only one operator, i.e. every time a rewrite rule is applied, the length of the total expression is reduced by one. This means that for any finite sequence $\Phi_1\Phi_2 ... \Phi_n$ where $\Phi_i \in \{N_a, G_b, F_c\}$ we can apply the rewrite rules in a systematic way (for example by always applying them on the first two operators in the sequence) and this process will terminate in *n-1* steps resulting in only one remaining operator (N_a, G_b, F_c or ε).

Therefore, we have defined a procedure that solves the sub problem of simplifying sequences of state operators to a single operator.

As an example, imagine that one persistent mapping (that uses the G operator) is defined in state 3, and another such mapping is defined in state 7. Then if we want to combine both

these mappings, such a new mappings is persistent from state 7, because
$G_3G_7 \Rightarrow G_{\max(3,7)} = G_7$.

4.8 Rewrite Rules for expressions with quantifiers

We will now describe the second step of the mechanical composition procedure that we have outlined in §4.5.1. Let us therefore investigate how expressions using the two quantifier operators are composed (as we precisely defined "quantifier operators" in section §4.5.1, not quantifiers in general) into one. The short notation is defined in the following way:

$\Phi\Lambda \Rightarrow \Theta$, where $\Phi,\Theta,\Lambda \in \{\alpha_1,\alpha_2\}$

Now we can write the rewrite rules for all four possible cases:

$$\alpha_1\alpha_1 \Rightarrow \alpha_1,\ \alpha_1\alpha_2 \Rightarrow \alpha_2,\ \alpha_2\alpha_1 \Rightarrow \alpha_2,\ \alpha_2\alpha_2 \Rightarrow \varepsilon_\alpha$$

We should remember that this notation hides the fact that these α :s actually are not identical copies because they could operate on different variables or different Boolean functions – this will be clear later in this section.
The ε_α operator means that neither α_1 or α_2 can be inferred.

Let us look for example at this particular rewrite rule:
$\alpha_1\alpha_1 \Rightarrow \alpha_1$

This mechanical application of the rewrite rule corresponds to this reasoning step (in §8.1.2.2 we use a related proof pattern):

$$\begin{aligned}
&\alpha_1(\lambda_1(x_i,y_j)) \wedge \alpha_1(\lambda_1'(y_j,z_k)) = \\
&\forall x_i y_j (\mathrm{Rel}\left(x_i,y_j\right) \to (\lambda_1(x_i,y_j))) \wedge \forall y_j z_k (\mathrm{Rel}\left(y_j,z_k\right) \to (\lambda_1'(y_j,z_k))) \Rightarrow \\
&\forall x_i z_k (\mathrm{Rel}\left(x_i,z_k\right) \to (\lambda_1(x_i,y_j) \wedge \lambda_1'(y_j,z_k))) = \\
&\alpha_1(\lambda_1''(x_i,z_k))
\end{aligned} \qquad \textbf{(4)}$$

(where $\lambda_1'' = \lambda_1 \circ \lambda_1'$)

So instead of doing this reasoning every time this situation occurs, we instead use the mechanical rewrite rules.

Above we described the mechanical application of the rewrite rules. The formal meaning of these is the following:

$$\Phi(\lambda(x_i, y_j)) \wedge \Lambda(\lambda'(y_j, z_k)) \rightarrow \Theta(\lambda''(x_i, z_k))$$

Where $\Phi \in \{\alpha_1(\lambda), \alpha_2(\lambda)\}$, $\Lambda \in \{\alpha_1(\lambda'), \alpha_2(\lambda')\}$, $\Theta \in \{\alpha_1(\lambda''), \alpha_2(\lambda'')\}$, λ is a Boolean function with the arguments (x_i, y_j) and λ' a function with the arguments (y_j, z_k) whereas λ'' has the arguments (x_i, z_j).

E.g., the meaning of α_1 (if it appears as the first symbol in the composition, i.e. Φ) is $\alpha_1(\lambda(x_i, y_j)) = \forall x_i y_j (R(x_i, y_j) \rightarrow \lambda(x_i, y_j)))$

E.g., the meaning of α_2 (if it appears as the second symbol in the composition, i.e. Λ) is $\alpha_2(\lambda(y_j, z_k)) = \exists y_j, z_k (R(y_j, z_k) \wedge (\lambda(y_j, z_k))$

Again, we should note that when two operators are combined (in an arbitrary order) they result in only one operator, i.e. every time a rewrite rule is applied, the length of the total expression is reduced by one. This means that for any finite sequence $\Phi_1\Phi_2...\Phi_n$ where $\Phi_i \in \{\alpha_1, \alpha_2\}$ we can apply the rewrite rules in a systematic way (for example by always applying them on the first two operators in the sequence) and this process will terminate in *n-1* steps resulting in only one remaining operator (α_1, α_2 or ε_α). Therefore, we have defined a procedure that solves the sub-problem of simplifying sequences of state operators to a single operator.

As an example, imagine that we have the two following mappings: COR[lemon$_1$,yellow_fruit$_2$] and COR[yellow_fruit$_2$, sour_fruit$_3$] and want to combine them in a state where they both exist. We call this the fruit example. Let us only look at the expressions within the state operators for both these mappings expressed in computation form in section 4.5.2 (i.e. we disregard the state operator expressions) and how these two will be combined. This will give us a sub-expression that matches one of the rules, and looks like this:

$$\alpha_1\alpha_1 \Rightarrow \alpha_1$$

and this mechanical application of the rewrite rule corresponds to equation 4 in this section.

The composition $\lambda_1'' = \lambda_1 \circ \lambda_1'$ (in equation 4) actually combines Boolean function (as exemplified in §4.5.2 for the COR.-mappings, i.e. the one used in the fruit example, and as will be defined in general in §4.9), so now we will investigate how these Boolean functions are described and combined (and this example will continue).

4.9 Representation and combination of Boolean functions

We will now describe the third step of the mechanical procedure that we have outlined in §4.5.1. We look now at functions that take two Boolean variables and constraints which of the four possible combinations of truth assignments that are allowed.

Let us now again describe the general form of the Boolean function (that we described in §4.5.1 but in a somewhat different form):

$$f(e_1,e_2) = (e_1 \wedge e_2 \wedge t_1) \vee (e_1 \wedge \neg e_2 \wedge t_2) \vee (\neg e_1 \wedge e_2 \wedge t_3) \vee (\neg e_1 \wedge \neg e_2 \wedge t_4)$$

where t_1, t_2, t_3 and t_4 are the Boolean parameters that decide how the function look like. Every parameter is either true or false, so a particular instantiation of the parameters will generate one of the 16 possible Boolean functions. Therefore, a tuple containing these parameters can be used to uniquely identify the corresponding Boolean function.

The use cases we want to cover will satisfy this convolution:

$$f(e_1,e_2) \wedge f(e_2,e_3) \Rightarrow f(e_1,e_3)$$

The Boolean function use disjunctive normal form (2-DNF) here.
Assuming that there is a second Boolean function of this form

$$f'(e_2,e_3) = (e_2 \wedge e_3 \wedge t_1') \vee (e_2 \wedge \neg e_3 \wedge t_2') \vee (\neg e_2 \wedge e_3 \wedge t_3') \vee (\neg e_2 \wedge \neg e_3 \wedge t_4')$$

we can combine the functions in the following way and create a new Boolean function $g(e_1,e_3)$:

$$g(e_1,e_3) = f(e_1,e_2) \wedge f'(e_2,e_3) \Rightarrow \\ (e_1 \wedge e_3 \wedge s_1) \vee (e_1 \wedge \neg e_3 \wedge s_2) \vee (\neg e_1 \wedge e_3 \wedge s_3) \vee (\neg e_1 \wedge \neg e_3 \wedge s_4) \quad \textbf{(5)}$$

All the parameters s_i are calculated like this:

$$s_1 = (t_1 \wedge t_1') \vee (t_2 \wedge t_3') \\ s_2 = (t_1 \wedge t_2') \vee (t_2 \wedge t_4')$$

$$s_3 = (t_3 \wedge t_1') \vee (t_4 \wedge t_3') \\ s_4 = (t_3 \wedge t_2') \vee (t_4 \wedge t_4')$$

Now we can continue the fruit example where we want to combine COR[lemon$_1$,yellow_fruit$_2$] and COR[yellow_fruit$_2$, sour$_3$], and look at the calculation of a sub expression. The latest was the following, and now we know what the Boolean functions λ_1=f$_1$ and λ_1'=f$_2$ actually are.

From (4) in section 4.7 we know that:

$$\forall x_1 y_2 (\mathrm{Rel}(x_1,y_2) \rightarrow (f_1(x_1,y_2))) \wedge \forall y_2 z_3 (\mathrm{Rel}(y_2,z_3) \rightarrow (f_2(y_2,z_3))) =$$

$$\forall x_1 y_2 (\mathrm{Rel}(x_1,y_2) \rightarrow ((lemon_1(x_1) \wedge yellow\_fruit_2(y_2)) \vee \\ (\neg lemon_1(x_1) \wedge \neg yellow\_fruit_2(y_2)))) \wedge \\ \forall y_2 z_3 (\mathrm{Rel}(y_2,z_3) \rightarrow ((yellow\_fruit_2(y_2) \wedge sour\_fruit_3(z_3)) \vee \\ (\neg yellow\_fruit_2(y_2) \wedge \neg sour\_fruit_3(z_3)))) \Rightarrow$$

$$\forall x_1 z_3 (\mathrm{Rel}(x_1,z_3) \rightarrow f_3(x_1,z_3)) \Rightarrow$$

$$\forall x_1 z_3 (\mathrm{Rel}(x_1,z_3) \rightarrow (lemon_1(x_1) \wedge sour\_fruit_3(z_3)) \vee \\ (\neg lemon_1(x_1) \wedge \neg sour\_fruit_3(z_3)))$$

We have managed to combine both the expressions with quantification over states and the Boolean function, and in the next section this example will be finalised.

4.10 Composing ontology mappings by using all kinds of rewrite rules

Because we have defined the rewrite rules for states operators and for quantifier operators, and the combination rules for Boolean functions, we can now combine all these and use them for combining two ontology mappings (we refer to the notation defined in §4.5):

$$
\begin{aligned}
&m_i \circ m_{i+1} = \\
&m_i(C_{1j}, C_{2k}) \wedge m_{i+1}(C_{2k}, C_{3l}) = \\
&op(\alpha(f(C_{1j}, C_{2k})))op'(\alpha'(f'(C_{2k}, C_{3l}))) \Rightarrow \\
&op''(\alpha''(f''(C_{1j}, C_{3l})))
\end{aligned}
$$

This is actually a form of composition of functions in the mathematical sense.
The algorithmic aspect of this procedure will be clearer using this notation:

$$
\begin{aligned}
&m_i(C_{1j}, C_{2k}) \wedge m_{i+1}(C_{2k}, C_{3l}) = \\
&op(\alpha(f(C_{1j}, C_{2k}))) \wedge op'(\alpha'(f'(C_{2k}, C_{3l}))) \overset{1}{\Rightarrow} \\
&op''(\alpha(f(C_{1j}, C_{2k})) \wedge \alpha'(f'(C_{2k}, C_{3l}))) \overset{2}{\Rightarrow} \\
&op''(\alpha''(f(C_{1j}, C_{2k}) \wedge f'(C_{2k}, C_{3l}))) \overset{3}{\Rightarrow} \\
&op''(\alpha''(f''(C_{1j}, C_{3l})))
\end{aligned}
$$

The first transformation (1) is the application of “rewrite rules for state operators” in a way that combines two state operators into one. This procedure is described in section 4.7. The second transformation (2) is the application of “rewrite rules for expressions with quantifiers” in a way that combines two operators into one. This procedure is described in section 4.8. The third transformation (3) is the application of “combination of Boolean functions” in a way two combines to such functions into one. This procedure is described in section 4.9.

To finish our fruit example we now have:

(here $e_1 = Lemon_1(x_1),\ e_2 = Yellow\_fruit_2(x_2),\ e_3 = Sour\_fruit_3(x_3)$, i.e. these are the predicates that correspond to the concept names)

$$\begin{aligned}
&\text{COR[lemon}_1\text{,yellow\_fruit}_2\text{] and COR[yellow\_fruit}_2\text{, sour\_fruit}_3\text{]} \Rightarrow \\
&(N_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_2\rangle,\langle \neg e_1 \wedge \neg e_2\rangle\}) \wedge \\
&(N_r\alpha_1\{\langle e_2 \wedge e_3\rangle,\langle \neg e_2 \wedge \neg e_3\rangle\} \wedge G_r\alpha_1\{\langle e_2 \wedge e_3\rangle,\langle \neg e_2 \wedge \neg e_3\rangle\}) \overset{1}{\Rightarrow} \\
&N_r\alpha_1\alpha_1\{...\} \wedge G_r\alpha_1\alpha_1\{...\} \overset{2}{\Rightarrow}
\end{aligned}$$

$$\begin{aligned}
&N_r\alpha_1\{((e_1 \wedge e_2) \vee (\neg e_1 \wedge \neg e_2)) \wedge ((e_2 \wedge e_3) \vee (\neg e_2 \wedge \neg e_3))\} \wedge \\
&G_r\alpha_1\{((e_1 \wedge e_2) \vee (\neg e_1 \wedge \neg e_2)) \wedge ((e_2 \wedge e_3) \vee (\neg e_2 \wedge \neg e_3))\} \overset{3}{\Rightarrow} \\
&N_r\alpha_1\{\langle e_1 \wedge e_3\rangle,\langle \neg e_1 \wedge \neg e_3\rangle\} \wedge G_r\alpha_1\{\langle e_1 \wedge e_3\rangle,\langle \neg e_1 \wedge \neg e_3\rangle\} = \\
&N_r\alpha_1\{\langle \text{lemon}_1(x_1) \wedge \text{sour\_fruit}_3(z_3)\rangle,\langle \neg\text{lemon}_1(x_1) \wedge \neg\text{sour\_fruit}_3(z_3)\rangle\} \wedge \\
&G_r\alpha_1\{\langle \text{lemon}_1(x_1) \wedge \text{sour\_fruit}_3(z_3)\rangle,\langle \neg\text{lemon}_1(x_1) \wedge \neg\text{sour\_fruit}_3(z_3)\rangle\} = \\
&\text{COR[lemon}_1\text{,sour\_fruit}_3\text{]}
\end{aligned}$$

Since we now can combine arbitrary adjacent ontology mappings, then let us now investigate a little bit more how we can reason about whether a new proposed mapping is consistent with a set of existing ones.

4.11 Defining the effect of negation

In the proof procedures that we will describe we will try to refute the negation of a statement.
We therefore must define what happens when we negate a relationship/mapping (and its computational constituents) and therefore we should define how the negation symbol affects any relationship/mapping. Firstly, we define how it affects the state operators.

The following rewrite rules will not explicitly be used later, but we use to infer the negation of various relations (see the end of this section):

$$\neg N(exp) \Rightarrow \varepsilon$$

$$\neg G(exp) \Rightarrow F(\neg exp)$$

$$\neg F(exp) \Rightarrow N(\neg exp) \wedge G(\neg exp)$$

Secondly, we define how the rewrite rules work for the quantifier operators. We infer this from the definitions.

$$\alpha_1(\lambda_1(x_i, y_j)) = \forall x_i y_j (R(x_i, y_j) \to (\lambda(x_i, y_j)))$$

$$\neg\alpha_1(\lambda_1(x_i, y_j)) = \neg\forall x_i, y_j (R(x_i, y_j) \to (\lambda_1(x_i, y_j)) = \exists x_i, y_j (R(x_i, y_j) \wedge (\neg\lambda_1(x_i, y_j))) = \alpha_2(\neg\lambda_1(x_i, y_j))$$
$$\neg\alpha_2(\lambda_1(x_i, y_j)) = \neg\exists x_i, y_j (R(x_i, y_j) \wedge (\lambda_1(x_i, y_j))) = \forall x_i, y_j (\neg R(x_i, y_j) \vee \neg(\lambda_1(x_i, y_j)) =$$
$$\forall x_i, y_j (R(x_i, y_j) \to (\neg\lambda_1(x_i, y_j)) = \alpha_1(\neg\lambda_1(x_i, y_j))$$

Therefore, the rewrite rules are the following.

$$\neg\alpha_1(\lambda_1(x_i, y_j)) \Rightarrow \alpha_2(\neg\lambda_1(x_i, y_j))$$
$$\neg\alpha_2(\lambda_1(x_i, y_j)) \Rightarrow \alpha_1(\neg\lambda_1(x_i, y_j))$$

Finally, we define negation for Boolean functions. We describe the effects of negation for the case of Boolean function with two variables.

$$\begin{aligned}
&\neg f(e_1, e_2)\\
&= \neg((e_1 \wedge e_2 \wedge t_1) \vee (e_1 \wedge \neg e_2 \wedge t_2) \vee (\neg e_1 \wedge e_2 \wedge t_3) \vee (\neg e_1 \wedge \neg e_2 \wedge t_4))\\
&= (\neg e_1 \vee \neg e_2 \vee \neg t_1) \wedge (\neg e_1 \vee e_2 \vee \neg t_2) \wedge (e_1 \vee \neg e_2 \vee \neg t_3) \wedge (e_1 \vee e_2 \vee \neg t_4)\\
&= \ldots\\
&= f'(e_1, e_2)
\end{aligned}$$

The negation first creates a CNF-form very quickly and then the expression has to be translated back to DNF-form.
To conclude this section, we can now mechanically negate any of the proposed relationships/mapping by negating its three constituents (the state operator, the quantifier operator and the Boolean function).

E.g. now we can calculate:

$$\neg Is(A_i, B_j) =$$
$$\neg N(...) \vee \neg G(\alpha_1((e_1 \wedge e_2) \vee (\neg e_1 \wedge \neg e_2) \vee (\neg e_1 \wedge e_2))) =$$
$$F\alpha_2(e_1 \wedge \neg e_2) =$$

Compatible (A_i, $\neg B_j$)

4.12 A reasoning procedure

4.12.1 The elementary steps used by the two reasoning procedures

We will describe the proof procedures here, whereas a more theoretical analysis of them will be provided in §8.2. Our approach is to use a reasoning method that creates a proof search tree by using breath-first search. The root of the tree (it is visualised in the following figures as being at the top) contains the negation of the newly proposed mapping, every *arc* corresponds to an applicable rule and every *node* is the result of applying that rule to the expression above.
The leaves are the nodes yet without children of the proof search tree during the reasoning procedure.

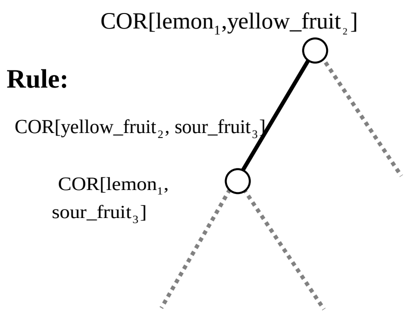

COR[lemon$_1$,yellow_fruit$_2$]
Rule:
COR[yellow_fruit$_2$, sour_fruit$_3$]
COR[lemon$_1$, sour_fruit$_3$]

Figure 9. If the rule is applied forward, the parent node is combined with the rule and the child node contains the result.

The reasoning procedure makes sure that no leaf can subsume another one leaf or node (i.e. so that unnecessary repetitions of proofs are removed).
In this work the rule is an existing mapping that has at least one common concept with the expression above. For example assume that COR[lemon$_1$,yellow_fruit$_2$] is the expression that the reasoning starts from, and that one of the “rules” is COR[yellow_fruit$_2$, sour_fruit$_3$]. Then it can actually be applied because the expression and the rule have (at least) one concept in common, namely the concept *yellow_fruit*$_2$.

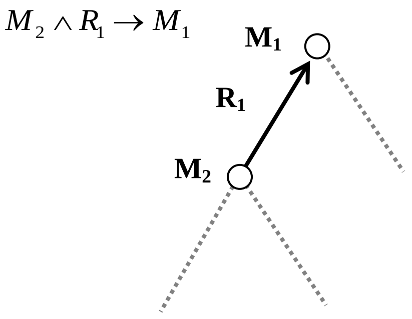


Figure 10. The general pattern, for how a mapping at one node, a rule and mapping at another node, are related.

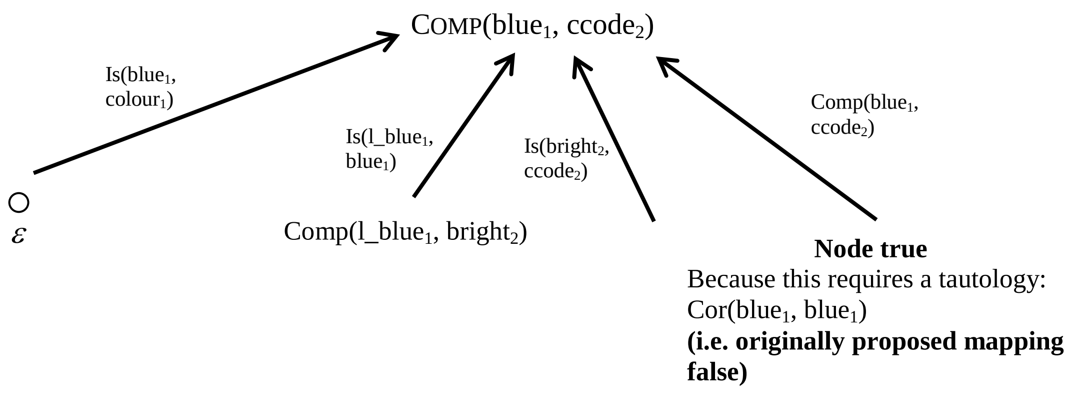


Figure 11. The purpose of this proof search tree is to find a proof for the refutation of Disjoint(blue$_1$, ccode$_2$), i.e. Comp(blue$_1$, ccode$_2$), and a refutation is found, because there is a node that we can prove is true.

Then this rule annotates the arc, and the child node will be the result of an application of the rule to the expression above, i.e. $\mathrm{COR}[\mathrm{lemon}_1, \mathrm{sour\_fruit}_3]$. In this example we are applying the rules forward, but they can also be applied backwards (see section 4.12.3).

At the leaves of a proof search tree, we would like see expressions like COR[x_i, x_i] (that is always true) or DISJOINT[x_i, x_i] (that is always false), because they terminate the search for a proof (depending on if we are looking a tautology or self-contradiction, we will see that this is different for the two algorithms that we defined, Algorithm A and Algorithm B).

When a new ontology mapping is proposed, there are two different kinds of proofs that can be made: such ones that apply the rules forward, as such that apply them backwards.

4.12.2 Comparison with first-order resolution

The standard first-order resolution procedure as defined by (Brachman, 2004, page 58) is similar to our procedure but the differences are

1. Our language is much less expressive.
2. They generate a resolution derivation where two clauses are combined into a new clause using binary resolution, and graphically they visualize this that the clauses are two nodes and their result (using binary resolution) is placed in their child node.
3. Every clause in their propositional resolution derivation is in CNF form, whereas in our case every clause is in the form specified in section §4.5.1.
4. They describe how the two quantifiers is standard FOL are eliminated, but in our case our procedure has to combine state symbols (described in section §4.7) and quantifier operators that quantify over various contexts (described in section §4.8) – so both these are "non-standard" operations.

Algorithm A

“Would adding G have introduced a contradiction in sphere sp of degree d_c that is higher than the defined degree p_c in that sphere?”

(tot is the estimated maximal size that sp can reach, p_c be level of consistency within this sphere of consistency sp)

```
INPUTS: G, sp, $d_r$, $p_r$, tot

l←0 (l is the current tree depth)
d← $(2-p_c)(tot-1)+2$
if G=IS|IS2|COR
   then return false
Root of the tree ← Negation of G
if contradiction(e, G)
    then return true

do loop {
  l←l+1
  if l>=d
      then return false
  At level l in the proof search tree:
     for every node $M_1$ at that level
     do Construct the list of allowed rules for $M_1$ (i.e. adjacent rels.)
            and call them $r_1, r_2, \ldots, r_n$
        for i←1 to n
            do construct a list P of elements e (that are allowed  mappings) that
                 satisfy the constraint: $e \wedge r_i \Rightarrow M_1$
                 Remove all elements from P subsumed by existing search tree  nodes
                 if P is empty
                     then, close this branch with a $\varepsilon$ and continue loop
                 else if any element e in P is self-evidently true
                          then the algorithm returns true
                       else Choose the most general element e, and
                          let it become a new node at level l+1
   if all branches for all nodes at this level are $\varepsilon$
        then the algorithm returns false
}
```

Figure 12. Algorithm A.

5. Their procedure terminates when the empty clause is encountered whereas in our case the termination is due to encountering a tautology or knowledge that is trivially true (Algorithm A) or trivially false (Algorithm B).

4.12.3 The proof procedure for the first kind of proof

Let us first take a look at the first proof procedure, and it is characterized by applying the rules “backwards” (see Figure 10). This is done by applying the combination procedure from section 4.10, but backwards, because we already have M_1 and R_1 but M_2 has to be generated. It is clear that this process is underspecified and would be nondeterministic if it we left it in this manner. Therefore, the algorithm has to choose the most general M_2 that satisfy $M_2 \wedge R_1 \rightarrow M_1$ in the case when several cases M_2 satisfy this equation.

It seems appropriate to first try to use breadth-first search, because the branching degree is rather limited. Algorithm A describes how this problem is solved. The reason why one of the lines reads
"If l>=d then return false" is that by then the algorithm has verified there are no proofs of size d-1, i.e. because we had decided that that we are not interested in contradiction proofs that can only be found if more computations would have been done.

Now we will try to solve the second case in the first example mentioned (see Figure 6) by allowing Algorithm A to create a proof search tree, i.e. testing if DISJOINT(blue$_1$, ccode$_2$) is consistent with the existing mappings. In the top node we put the negation of the newly proposed mapping, i.e. COMP(blue$_1$, ccode$_2$). The information in the nodes/leaves arises when the algorithm traverses the knowledge in Figure 6.
If we look at Figure 11 we see that the tree has four different branches on the first level. It is interesting to know that if would have evaluated the result for all six branches then some of them would actually lead to ε (i.e. there is no M_2 that satisfies $M_2 \wedge R_1 \rightarrow M_1$) and that means we wouldn’t have continued exploring these branches. Moreover, if the result of one branch subsumes the result of another, it is enough to continue investigating the more specific mapping. In our example, the depth of the proof is two, and that corresponds to closed sequence of mappings (a “loop”) consisting of two mappings. In general, if there is a shortest closed mapping sequence (“a loop”) of length *n*, then the proof depth will be the same. We will see in chapter 8 why only relations of type COMPATIBLE or DISJOINT can cause a contradiction using the current reasoning system, so that is why the line " If G=IS|IS2|COR return false" is in the algorithm A.

Let us now investigate correctness and completeness for this particular case when the negation of a newly proposed mapping G can inferred from a minimal set of existing mappings R_n, R_{n-1},…,R_1. We investigate the case when the level of consistency within a sphere of consistency is not bound, $p_c=1$. This is illustrated in Figure 13. The existing mappings are used as rules. If we go back to the definition of the rule application procedure we see that

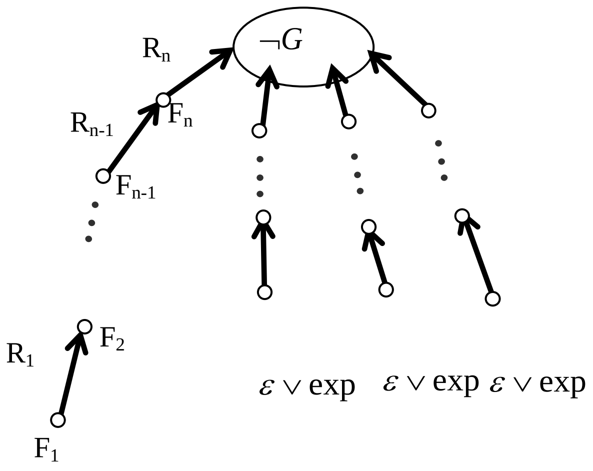


Figure 13. The newly proposed mappings is called G and R_n, R_{n-1},..., R_1 are the mappings that are used as rules in this proof tree, where F_1 is true.

$$F_1 \wedge R_1 \to F_2$$

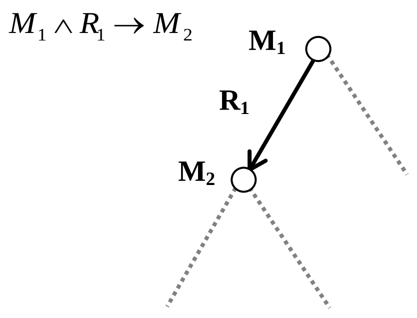
$$F_1 \wedge R_1 \wedge R_2 \to F_3 \text{ (because } F_2 \wedge R_2 \to F_3\text{)}$$

.

.

$$F_1 \wedge \bigwedge_{i=1..n} R_i \to \neg G$$

From the point of the view of the algorithm, it is doing breadth-first search and creating a proof search tree. If all paths except one end with an ε or any expression that still will be processed (this is called exp in Figure 13), then that single path stops at an expression F_1 that we can prove is true (because it is a tautology or follows from one of the existing mappings). Then, from a logical point of view, we have a set of mappings whose conjunction implies the negation of a proposed mapping, i.e.



$$\bigwedge_{i=1..n} R_i \to \neg G$$

Figure 14. The rules are applied in a different way this time.

Algorithm A is sound and complete (assuming that the composition operation in every step is sound, and that will be demonstrated in Chapter 8). Also, if there are several different proofs (i.e. sets of existing mappings) then it will find the shortest proof (i.e. smallest set). Again, this only holds for $p_c=1$.

Algorithm B
"Would adding G have introduced a redundancy in sphere sp of degree d_r that is higher than the defined degree p_r in that sphere?"
(tot is the estimated maximal size that sp can reach, p_r is the level of redundancy within this sphere of consistency sp)

```
INPUTS: G, sp, $d_r$, $p_r$, tot

l←0 (l is the current tree depth)
d←$(2-p_r)$(tot-1)+2
Root of the tree ← negation of the newly proposed mapping G

do loop {
 l←l+1
 if l>=d
     then return false
 At level l in the search tree:
 For every node $M_1$ at that level
    if $M_1$ already existed at any lower level in the tree
       then close branch with $\varepsilon$ and continue (node) loop
    Construct the list of allowed rules for $M_1$, (i.e. adjacent rels.)
    and call them $r_1, r_2, \ldots, r_n$
    for i← i=1 to n
       do $M_1 \wedge r_i$ =>e
          if e=$\varepsilon$
             close branch and continue loop
         else
            if e is self-contradictory
               then the algorithm returns true
            else
               let e become a new node at level l+1
 if all branches for all nodes at this level are $\varepsilon$
    then the algorithm returns false
}
```

Figure 15. Algorithm B.

4.12.4 The proof procedure for the second kind of proof

Let us now take a look at the second proof

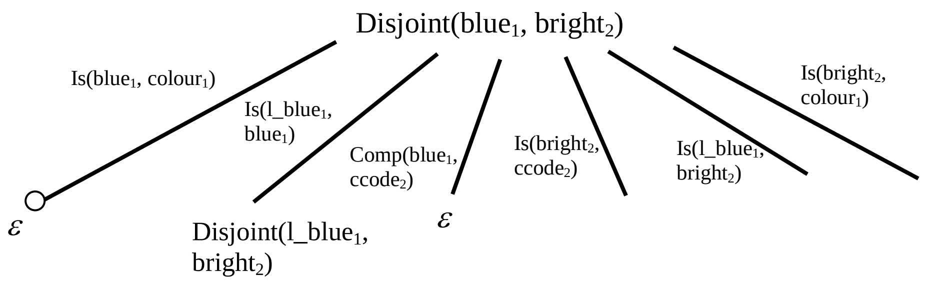

Disjoint($blue_1$, $bright_2$)
Is($blue_1$, $colour_1$)
Is(l_blue_1, $blue_1$)
Comp($blue_1$, $ccode_2$)
Is($bright_2$, $ccode_2$)
Is(l_blue_1, $bright_2$)
Is($bright_2$, $colour_1$)
ε
Disjoint(l_blue_1, $bright_2$)
ε

Figure 16. purpose of this proof search tree is to find a refutation proof for Comp($blue_1$, $bright_2$), and some of the values of the nodes and edges are shown. This proof search tree will eventually terminate because all leaves will at some point have epsilons.

procedure, and it is characterized by applying the rules "forwards" (see Figure 14). Algorithm B describes how this problem is solved. The reason why one of the lines reads "If l>=d then return false" is that by then the algorithm has verified there are no redundancy proofs of size d-1, i.e. because we had decided that that we are not interested in redundancy proofs that can only be found if more computations would have been done.
Now we will try to solve the first case in the first problem (investigating if COMP($blue_1$, $bright_2$)) can be inferred from the existing mappings (i.e. create a redundancy if it is added), but we will investigate how a refutation proof looks like where the rules as applied forward. Then we simply take the proposed mapping and negate it (the negation is in fact DISJOINT($blue_1$, $bright_2$)) and we are allowed to apply the same kind of rules as before. If we reach a node that we know is false, then the proposed mapping is consistent with the existing ones (because we have refused the refutation) and it was in fact already known, i.e. we have proved it is true. Let us now investigate what happens when we try to find a proof for the refutation of COMP($blue_1$, $bright_2$).

The beginning of the creation of this proof search tree is illustrated in Figure 16, and in this case the process will never find a leaf that can be proven to be false, so the algorithm will terminate when all leaves contain epsilon expressions and the answer of the algorithm is "no" – i.e. COMP($blue_1$, $bright_2$) cannot be inferred from existing ontologies or mappings within this sphere of consistency.

Let us now investigate correctness and completeness for this particular case when a newly proposed mapping G can be inferred from a minimal set of existing mappings R_n, R_{n-1},…,R_1 or is contradicting a set of existing mappings (there is a third case that will be investigated in the next section). This is illustrated in Figure 17. The existing mappings are used as rules. If we go back to the definition of the rule application procedure we see that:

$$\neg G \wedge R_n \rightarrow F_n$$
$$\neg G \wedge R_n \wedge R_{n-1} \rightarrow F_{n-1} \text{ (because } F_n \wedge R_{n-1} \rightarrow F_{n-1}\text{)}$$
.
.
$$\neg G \wedge \bigwedge_{i=1..n} R_i \rightarrow F_1$$

From the point of the view of the algorithm, it is doing breadth-first search and creating a proof search tree. If all paths except one end with an ε expression (or are in the process of being investigated - denoted as "exp" in figure 8), then these paths terminate (or would still have been investigated) and that single path stops at an expression F_1 that we can prove is a contradiction (because it is a negated tautology or contradicts one of the existing mappings). Then, from a logical point of view, we have the following (when F_1 is false and the arrow symbolizes a sequence of logical consequences):

$$\neg G \wedge \bigwedge_{i=1..n} R_i \Rightarrow false$$
$$\neg(\neg G \wedge \bigwedge_{i=1..n} R_i)$$
$$G \vee \neg \bigwedge_{i=1..n} R_i \Rightarrow$$
$$\bigwedge_{i=1..n} R_i \rightarrow G$$

The procedure is sound (assuming the composition at every step is correct, this will be investigated in Chapter 8).

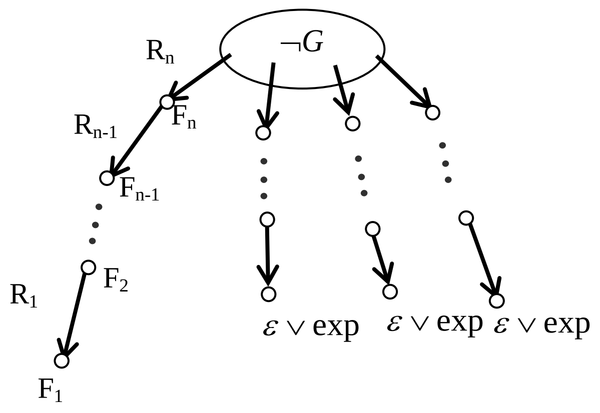


Figure 17. The newly proposed mapping is called G and is negated, and R_n, R_{n-1},…,R_1 are the mappings that are used as rules in this proof search tree, where F_1 has to be false.

The final case we will investigate is when both algorithms answer no. We must first investigate if both algorithms could have answered yes. If that happens, then that is an example of unsoundness, and when p_c=1 then that can never happen (assuming the soundness of algorithms A and B). Also, we will see that the rules in chapter 5 will investigate first run algorithm A (checking for inconsistency) and if it returns true, then algorithm B won't run in the same sphere of consistency (in the policy that we will describe).

4.12.5 The third case

Finally we have to look at the third case, which requires most computations. In that case a proposed new relationship/mapping neither contradicts the existing mappings nor already can be inferred from them. We want to build a system that works in all three cases. The first obvious way to achieve this is given a new proposed mappings first run it in algorithm A, and then in algorithm B. If algorithm A returns “yes”, then the proposed mapping is inconsistent with the current mappings, and if it returns “no” we have to run algorithm B. If algorithm B returns “yes” then the mapping is valid and could already be inferred, and if it returns “no” then the mapping is consistent with the existing ones, but is not already known. This sequence of reasoning will be interleaved with communication with the relevant contexts as will be described in chapter 5 – e.g. if in a situation it is enough to know if a proposed change does introduce contradiction, then it might be not necessary to run Algorithm B.

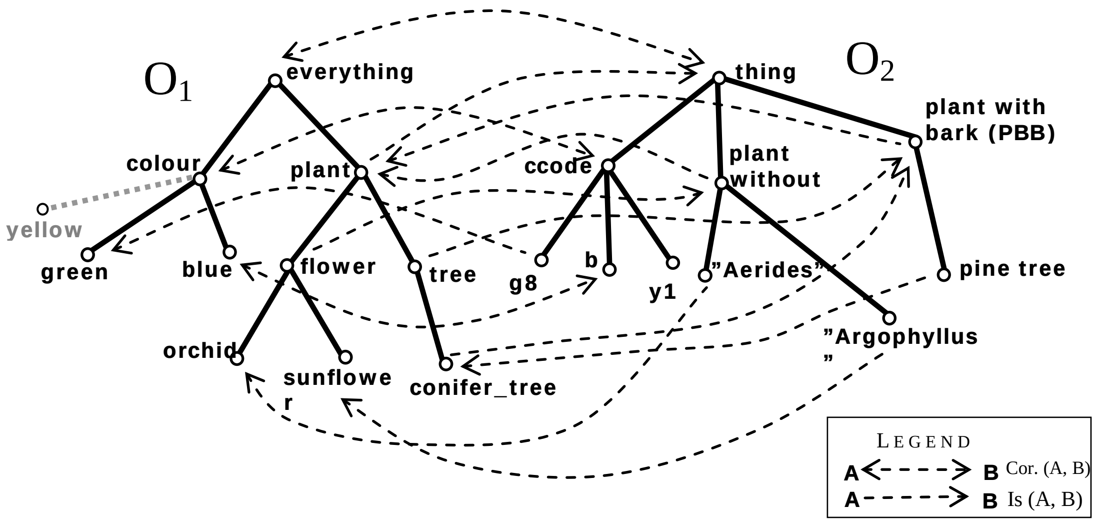


Figure 18. Here we see two ontologies and some mappings between them. We want to evolve the state of theses ontologies by creating a new concept *yellow*$_1$ and these two mappings: Is(yellow$_1$, colour$_1$) and Is(yellow$_1$, ccode$_2$), but we first we have to see if this change maintains introduces contradictions.

A consequence for the third case is that neither the first or second proof search tree will terminate because it encounters a node that is true or false, but they will terminate because the proofs will reach ε symbols in all open branches, so the search space has been completely explored. If Algorithm A and Algorithm B are sound, then this last procedure is also sound.

4.13 An example motivating a whole lifecycle in the knowledge infrastructure

In this new example we have two ontologies (see Figure 18) describing two different departments in a home garden store. For example, the first department is more directed towards customers and let us assume that some customers have asked questions involving the colour yellow – a concept that yet does not exist in that ontology. E.g. an ontology application running on top of the knowledge infrastructure might pass on the question "Do you have yellow sunflowers?" in a formal language and whereas "sunflower" is recognised as a formal concept, "yellow" is not so by responding to the query a proposal to evolve the ontology will be sent. As will be described in Chapter 5 processing that proposal will include finding out if it will create an inconsistency or redundancy in relevant sphere of consistency. So we want to evolve the whole system by inventing a new concept in ontology 1, namely $yellow_1$ and at the same time add two new links: Is(yellow$_1$, colour$_1$) and Is2(yellow$_1$, y1$_2$) (the interpretation of this is that y1 corresponds to a nuance of yellow). The first one describes the relationship of the concept within its home ontology, whereas the second link is mapping to another ontology. Let us first investigate if Is(yellow$_1$, colour$_1$) maintains the consistency of the system. We could run Algorithm A. However, we would never be to find contradiction, because yellow$_1$ only has one link to the remaining set of mappings and there is no sequence of mappings that begins with a concept, passes through yellow$_1$ and ends at its beginning (i.e. a "loop"). So in these cases we always accept the newly proposed mapping. Consequently, now Is(yellow$_1$, colour$_1$) is a part of the set, and it remains to investigate Is2(yellow$_1$, y1$_2$). If we run protocol rules described in Chapter 5 they will invoke algorithm A that will returns false, and then Algorithm B that also will return false. This means that the proposed relationship is new and protocol will communicate this to the affected contexts (the precise meaning of this will be defined in chapter 5).

Chapter 5 A description and formalisation of the protocol language of the infrastructure mechanism

5.1 Introduction

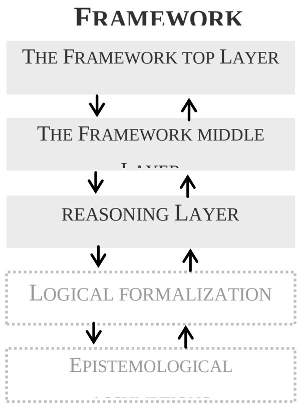


Figure 19. The framework and its layers.

We will describe an infrastructure mechanism that has semantic autonomy as we have defined it.
The solution is a framework consisting of five layers (see Figure 19). The two bottom layers represent the epistemological and logical assumptions (described in Chapter 3) whereas the three top layers constitute the executable system itself. We will now describe the whole framework and the protocol language. Because we are interested in automating the process of proposing and reconciling ontological changes we therefore define a rule-based process language that will formalise how proposals to evolve ontologies and mappings can be expressed and how they are managed.

5.2 Formal syntax of the protocol language

We first define all the syntactic categories of the vocabulary of the whole framework.The syntactic categories are:

i in *InfraMech*	The set of infrastructure mechanisms.
os in *Ostate*	This is the set of ontology states and a state only changes when the ontology is changed.
ns in *Nstate*	This is the set of negotiation states and such a state is only increased every time the protocol is processed one step.

D in *Dec*	This is a set of declarations of a procedures that have a head and body and can be executed.
c_i in *Context*	This is the set of all contexts as they are defined in chapter 3 and used in a computational way in chapter 4.
contexp in *ContExp*	This is the set of context expressions.
conv in *ConVal*	This is a set of sets of contexts.
op in *OntOp*	This is the set of ontology operations as defined in §5.3 that change ontologies.
spec_op in *SpecOntOp*	This is the set containing a special ontology operation as defined in §5.3.
cv in conVar	This is the set of concepts in the ontologies of all contexts.
ce in *KnoSet*	This is the set of sets of ontology relationships and mappings.
be in *BExp*	This is the set of Boolean expressions.
bv in *Bval*	This is the set of Boolean values
ent in *Ent*	This is the set of autonomous entities that exist that can be involved in initiating actions or processing initiated actions.
G in *GlobVar*	This is the set of globally defined Boolean predicates
BP in *FunVar*	This is the set of Boolean predicates.
CT in *TopCom*	This is the set of the top commands, i.e. the only ones that can be executed first.

CF in *FraCom*	This is the set of special framework commands that can change ontologies.
CM in *Com*	This is the set of general other commands.
CFM in *TopOrFCom*	This set is the union of *FraCom* and *Com*
spe in *SpExp*	This is the set of expressions that return spheres of consistency
spv in *SpFun*	This is a set of sets of spheres of consistency.
sp in *Sp*	This is the set of spheres of consistency.
S in *var*	This is the union of *spv* and the set of sets of integers.
m in *Mappings*	This is the set of the five ontology mappings described in §3.3.2.
r in *Ontorel*	This is the set of relationships within an ontology described in §3.3.1.
FN in *Fun*	This is the set of functions.
message in *Strings*	This is the set of strings.

This protocol has been inspired by (Robertson 2004) but our formalism does not use message-passing, except when MCHOICE communicates something to contexts that then have to do a decision. The whole system S consists of *n* different local contexts c_i (i=1…n), their ontologies and the mappings between them. Every local context possesses its own ontology.

Here follows definitions of the abstract syntax of all the layers in the whole framework.

The definitions are:

$i ::= D$

$D ::= CT \Leftarrow CM \mid CT \Leftarrow CM, D' \mid CF \Leftarrow CM, D'$

$ent ::= c_i \mid F$

$be ::= ent : BP \mid G \mid be' \vee be'' \mid be' \wedge be'' \mid True \mid False$

$op ::=$ add_mapping(m, c_j, c_k) $\mid$ add_ontorel(r, c_j) $\mid$ delete_mapping(m, c_j, c_k) $\mid$ delete_ontorel(r, c_j) $\mid \varepsilon$

$spec\_op ::= RC(x,\ ce)$

$G ::= Newconcept(cv)$

$BP ::= C\_CONTRA(spe,\ op) \mid IS\_INFERABLE(spe,\ op) \mid IS\_NEW(spe,\ op)$

$\quad \mid CONFIRM(op) \mid REFUSE(op)$

$FN ::= CREASON(sp,\ ont\_op)$

$contexp ::= CONTEXTS\_OF(spe)$

$spe ::= Spheres(c_i, c_j) \mid Spheres(c_i) \mid x$

$CT ::= c_i : PROPOSE(op)$

$CF ::= F : DO(op)$

$CM ::= F : MCHOICE(contexp, message, CFM,\ CFM) \mid F : DO(spec\_op) \mid$

$\quad F : REQUEST(CT) \mid be' \Rightarrow CM'$ or $be'' \Rightarrow CM'' \mid be \Rightarrow CFM'$

$CFM ::= CF \mid CM \mid CFM' \barwedge CFM''$

$\bigvee_{x \in S} be ::= be[v_1/x] \vee be[v_2/x] \vee \ldots \vee be[v_k/x]$ where S={$v_1, v_2, \ldots, v_k$}

$\bigwedge_{x \in S} be ::= be[v_1/x] \wedge be[v_2/x] \wedge \ldots \wedge be[v_k/x]$ where S={$v_1, v_2, \ldots, v_k$}

$\overline{\bigwedge}_{x \in S} CM ::= CM[v_1/x] \barwedge CM[v_2/x] \barwedge \ldots CM[v_k/x]$ where S={$v_1, v_2, \ldots, v_k$}

The informal descriptions of the meaning of syntax of the various layers and the actual protocol of our infrastructure mechanism will be presented in §5.3 (the reasoning layer), §5.4 (the framework middle layer) and §5.5 (the framework top layer). Section 5.6 will provide the formal computational semantics. We now just define the following symbols (some will be used in §5.6):

$\barwedge$ is a sequential composition of commands.

$Or(bv, bv')$ is the result of applying the Boolean function OR on the Boolean values bv and bv'.

$And(bv, bv')$ is the result of applying the Boolean function AND on the Boolean values bv and bv'.

5.3 The reasoning layer

Here, we repeat the functionality of the reasoning layer, because these expressions can be used as predicates by the protocol defined in §5.2.

The vocabulary of the reasoning layer contains the following predicates:

Predicates and parameters	**Meaning**
F: C_CONTRA(*sp*, *op*)	returns true if *op* would have introduced a contradiction in sphere *sp* of degree d_c that is higher than the defined degree p_c in that sphere.
F: IS_INFERABLE(*sp*, *op*)	returns true if *op* would have introduced a redundancy in sphere *sp* of degree d_r that is higher than the defined degree p_r in that sphere.
F: IS_NEW(*sp*, *op*)	returns true if neither F: c_contra(*sp*, *op*) or F: is_inferable(*sp*, *op*) are true.
F: CREASON(*sp*, *op*)	returns a subset of *sp* that creates a contradiction if *op* is performed.

The first 3 statements return either true or false, the fourth returns a subset. In the case when C_CONTRA(*S*, *op*) is true, CREASON(*S*, *op*) returns one of the contradiction reasons, i.e. one of the minimal subsets in the whole system S that show that S with the *op* performed creates a contradiction.

We also repeat the definition of the predicate Spheres(...).

Function	**Return values**
Spheres(c_i, c_j)	All spheres of consistency that contain *both* contexts *i* and *j*.
Spheres(c_i)	All spheres of consistency that contain context *i*.

The current list of operations that evolve ontologies or mappings between them is the following (and the second mentions a special ontology operation):

ont_op = add_mapping(m, c_j, c_k) | add_ontorel(m, c_j, d_j) |
delete_mapping(m, c_j, c_k) | delete_ontorel(m, c_j, d_j) | ε
spec_ont_op= RC(*sp, P*) [where P $\subseteq$ knowledge inside (*sp*)]

RC() is an operation that takes all the knowledge (ontology relations and mappings) in sphere *sp* and removes the inconsistent subset P. If there are several alternative such inconsistent subsets, several of them might have to be removed in order to make the whole sphere sp consistent, i.e. the operation RC() would have to be repeated.

5.4 The framework middle layer

This vocabulary will be used for defining this layer (and also some of the functionality of the reasoning layer):

Predicates and parameters	**Meaning**
c_i : CONFIRM (*op*)	c_i accepts that *op* should be performed
c_i : REFUSE (*op*)	c_i rejects the performance of *op*
F: DO (*op*)	The framework mechanism performs *op*
F:MCHOICE(*contexp*, message, {c_j, c_k, …}, *CFM₁*, *CFM₂*)	The framework mechanism chooses between the operations CFM$_1$ and *CFM$_2$* on behalf of several contexts.
contexts_of(*sp*)	A conjunction of all contexts included in the spheres *sp*.

The first rule at this level is the following.

$$\text{F: MCHOICE}(\textit{contexp}, \text{message}, \text{CFM}_1, \text{CFM}_2) \Leftarrow$$
$$\left(\left(\bigwedge_{c_x \in val(contexp)} \text{c}_x\text{: CONFIRM}(\text{CFM}_i)\right) \wedge i \in \{1,2\} \Rightarrow CFM_i\right) or$$
$$\left(i \in \not\{1,2\} \quad \wedge \bigvee_{c_x \in val(contexp)} \text{c}_x\text{: REFUSE}(\text{CFM}_i) \Rightarrow \bigvee_{c_x \in val(contexp)} \text{c}_x\text{: CONFIRM}(\text{CFM}_i) \qquad DO(\varepsilon)\right)$$

(6)

MCHOICE() is a procedure where a decision is made if a certain command (CFM_1) should be performed or another one (CFM_2). The entities involved in this decision procedure are one or more contexts. They are returned by the expression *contexp*. The formalization of the rule (above) says that if all involved local contexts choose one of the ontology operations, then that becomes their joint choice, and if there is some disagreement then the joint choice is to do nothing. We assume here that for any CFM_i appearing as a parameter in MCHOICE(), either CONFRIM(CFM_i) is true or REFUSE(CFM_i) is true, but this model does not include the decision procedure for how that choice is made. It is made by the contexts in *contexp* as a response to the contents of *message* and the contents of CFM_1 and CFM_2.
The DO() statement is important and can only be performed by the framework mechanism itself when it actually performs an ontology operation. **It is actually the DO() statement that moves the whole system S to the next state** – simply because it changes S. Before then, the system only does hypothetical reasoning ("what would have happen if an operation *op₁* were performed?").

We will now investigate what happens when a local context initiates a proposal to add a mapping to another local context.

Case 1. Proposing to add an ontology mapping.

$$\mathrm{c_j: \ PROPOSE(add\_mapping(m,\ c_j,\ c_k))} \Leftarrow$$

$$\left(\begin{array}{l} \left(\bigvee_{x\in spheres(c_j,c_k)} \mathrm{F{:}C\_CONTRA(x,\ add\_mapping(m,\ c_j,\ c_k))}\right) \Rightarrow \\ \mathrm{(F{:}\ MCHOICE(contexts\_of(}spheres(c_j,c_k)\mathrm{)),} \\ \mathrm{"contradicted:"+F{:}\ CREASON(x,\ add\_mapping(m,\ c_j,\ c_k)),} \\ \ DO(\varepsilon), \\ \quad \bigwedge_{x\in spheres(c_j,c_k)} \mathrm{F{:}C\_CONTRA(x,\ add\_mapping(m,\ c_j,\ c_k))} \Rightarrow \\ DO\mathrm{(RC(x,CREASON(x,\ add\_mapping(m,\ c_j,\ c_k)))))} \end{array}\right) or \qquad (7)$$

$$\left(\begin{array}{l} \left(\bigvee_{x\in spheres(c_j,c_k)} \mathrm{F{:}\ IS\_INFERABLE(x,\ add\_mapping(m,\ c_j,\ c_k))}\right) \Rightarrow \\ \mathrm{F{:}\ MCHOICE(\{c_j,\ c_k\},\ "already\_known",} DO(\varepsilon),\ DO\mathrm{(add\_mapping(m,\ c_j,\ c_k)))} \end{array}\right) or$$

$$\left(\begin{array}{l} (\bigwedge_{x \in spheres(c_j, c_k)} \text{F: IS\_NEW}(x, \text{add\_mapping}(m, c_j, c_k))) \Rightarrow \\ \text{F: MCHOICE}(\{c_j, c_k\}, \text{"new"}, DO(\varepsilon),\ DO(\text{add\_mapping}(m, c_j, c_k))) \end{array} \right)$$

This statement in the beginning of the rule above means: a local context c_j is proposing to add a mapping from its ontology to another local context c_k (see Figure 49 in §8.4.2 for a graphical visualization of this rule that facilitates its understanding). This rule uses some statements from the reasoning layer. Only one of the three sub-rules (inside this big rule) can actually be activated. Intuitively, this formalization (7) then says that if the proposed change would introduce a forbidden contradiction in *any* sphere containing both c_j and c_k, then one of the contradiction reasons is communicated to *all* of these affected spheres and *all* the contexts inside spheres containing both c_j and c_k will decide together to either do nothing or that one of the reasons for the contradiction is removed in every sphere where a contradiction occurs (but without adding the proposed mapping within the same step). Notice that all local contexts that have ontologies in spheres where one of the contradictions resides, have to participate in making this decision (this situation is referred to as "All involved inside the sphere..." below). The second and third sub-rule in (7) express that if the mapping can already be inferred or is new, the change is allowed but the two local contexts involved have to decide if they actually want to have it performed (this situation is referred to as "Pair" below). E.g. if, in Figure 18, we invent the concept **yellow** in O_1 and then propose to add Is(**yellow, colour**) then it will be classified as IS_NEW by the process above, because this proposal neither creates a contradiction nor redundancy.

Table 2. The table shows how decisions are made if c_i: propose (*ont_op*) is proposed.

***ont_op*=**	**c_contra() is true**	**is_inferable() is true**	**is_new() is true**
add_mapping(m, c_j, c_k)	All involved inside the sphere(s) of consistency	Pair	Pair
delete_mapping(m, c_j, c_k)	Contradiction cannot happen.	Pair (i.e. mappings was already deleted)	Pair (i.e. mapping existed)
add_ontorel(r, c_j)	All involved inside the sphere(s) of consistency	Individual	Individual

delete_ontorel(r, c_j)	Contradiction cannot happen.	Individual (i.e. relationship did not exist)	Individual (i.e. rel. existed)

Now we have investigated the case (case 1 above) when the proposal is to add a mapping between two ontologies, and the formalization showed what happens in the three cases. Table 2 above summarizes what this formalization would look like for the *four cases* when the proposal is to delete an ontology mapping, or add or delete an ontology relation. In all these cases formulas similar to (7) would be defined, but using the appropriate decision-making entities – this is done below. The term “individual” in the table means that the local context that created the proposal can decide itself if it wants the logically allowed change to *actually* be performed. If the proposal is to delete a mapping or delete a relation within an ontology this always logically allowed (because that is safe in this particular logic), so the individual local context or the pair of contexts decide whether to actually perform this act. This policy is allowing for individual ontologies that are not always singly connected.

Case 2. Proposing to delete an ontology mapping.

$$\mathrm{c_j:\ PROPOSE(delete\_mapping(m,\ c_j,\ c_k))} \Leftarrow$$

$$\left(\begin{array}{l}(\bigvee_{x \in spheres(c_j, c_k)} \mathrm{F:\ IS\_INFERABLE(x,\ delete\_mapping(m,\ c_j,\ c_k)))} \Rightarrow \\ \mathrm{F:\ COMM(\text{"already\_deleted"},\ \{c_j,\ c_k\})}\ \overline{\wedge} \\ DO(\varepsilon) \end{array}\right) or$$

$$\left(\begin{array}{l}(\bigwedge_{x \in spheres(c_j, c_k)} \mathrm{F:\ IS\_NEW(x,\ delete\_mapping(m,\ c_j,\ c_k)))} \Rightarrow \\ \mathrm{F:\ MCHOICE(\{c_j,\ c_k\},\ \text{"still\_exists"},} DO(\varepsilon),\ DO(\mathrm{delete\_mapping(m,\ c_j,\ c_k)))} \end{array}\right)$$

The second case investigates when a context c_j is proposing to delete a mapping from its ontology to another local context c_k. In the particular logic we are investigating, deleting a mapping cannot create a contradiction (but that is not universally true). Therefore, to simplify things, in this case the protocol firstly investigates if the deletion is inferable (i.e. the mapping does not exist) and in that case the only option is to do nothing, because deleting something that does not exist is indistinguishable from doing nothing. Secondly, if deleting the mapping would actually change the system (i.e. it still existed), then the two involved contexts (c_j and c_k) decide to either do nothing or actually delete the mapping.

Case 3. Proposing to add an ontology relationship.

$$
\begin{array}{l}
\mathrm{c_j:\ PROPOSE(add\_ontorel(r,\ c_j))} \Leftarrow \\
\left(\begin{array}{l}
(\bigvee_{x \in spheres(c_j)} \mathrm{F{:}C\_CONTRA(x,\ add\_ontorel(r,\ c_j)))} \Rightarrow \\
\mathrm{(F{:}\ MCHOICE(contexts\_of(spheres(c_j)),} \\
\text{"contradicted:"}\mathrm{+F{:}\ CREASON(x,\ add\_ontorel(r,\ c_j)),} \\
,\ DO(\varepsilon), \\
\quad \overline{\bigwedge}_{x \in spheres(c_j)} \mathrm{F{:}C\_CONTRA(x,\ add\_ontorel(r,\ c_j))} \Rightarrow \\
DO\mathrm{(RC(x,CREASON(x,\ add\_ontorel(r,\ c_j)))))}
\end{array}\right) or \qquad \textbf{(8)} \\
\left(\begin{array}{l}
(\bigvee_{x \in spheres(c_j)} \mathrm{F{:}\ IS\_INFERABLE(x,\ add\_ontorel(r,\ c_j)))} \Rightarrow \\
\mathrm{F{:}\ MCHOICE(c_j,\ \text{"already\_known"},} \\
DO(\varepsilon),\ DO\mathrm{(add\_ontorel(r,\ c_j)))} \\
\end{array}\right) or \\
\left(\begin{array}{l}
(\bigwedge_{x \in spheres(c_j)} \mathrm{F{:}\ IS\_NEW(x,\ add\_ontorel(r,\ c_j)))} \Rightarrow \\
\mathrm{F{:}\ MCHOICE(c_j,\ \text{"new"},} \\
DO(\varepsilon),\ DO\mathrm{(add\_ontorel(r,\ c_j)))} \\
\end{array}\right)
\end{array}
$$

This rule above takes care of the case when a context c_j proposes to add an ontology relationship r within itself. If that creates a contradiction in any of the spheres of consistency that c_j belongs to, then this is communicated to all of them including some contradicted knowledge within that sphere. Then, in the case of a contradiction all contexts in the spheres of consistency that contain c_j decide either to do nothing or that in every sphere that contain c_j and where a contradiction is made, one of the contradiction reasons are removed. In the case that the proposed ontology relationship is inferable this is communicated to all contexts in that sphere. However, the context c_j decides *itself* if it wants to do nothing or add the proposed ontology relationship. So this is the essence of a contextual ontology: decisions about changing it are made locally as long as this does not create any problems in the wider system, in contrast to ontology mappings that are governed by shared authority. If the proposed ontology relationship is new then that is communicated to all contexts within a sphere, but again, it is the context c_j itself that decides if it want to do nothing or actually add the proposed ontology relationship – because this change of knowledge wouldn't create any problems the final decision to do it or not is *localised*.

Case 4. Proposing to delete an ontology relationship.

$$\mathrm{c_j: \ PROPOSE(delete\_ontorel(r,\ c_j))} \Leftarrow$$

$$\left(\begin{array}{l} (\bigvee_{x \in spheres(c_j)} \mathrm{F: IS\_INFERABLE(x,\ delete\_ontorel(r,\ c_j))} \Rightarrow \\ \mathrm{F: COMM(\text{"already\_deleted"},\ c_j)} \ \overline{\wedge} \\ DO(\varepsilon) \\ \end{array} \right) \textit{or}$$

$$\left(\begin{array}{l} (\bigwedge_{x \in spheres(c_j)} \mathrm{F: IS\_NEW(x,\ delete\_ontorel(r,\ c_j))} \Rightarrow \\ \mathrm{F: MCHOICE(c_j,\ \text{"still\_exists"}}, DO(\varepsilon),\ \mathrm{delete\_ontorel(r,\ c_j))} \end{array} \right)$$

This final rule declaration describes that when a context proposes to delete a relationship in its ontology, then if that is already inferable then that is communicated to it and it can nothing do. But if the relationships actually exist and can be deleted then that context decides itself if it want to delete the relationship.

5.5 The framework top layer

We now formalize the top layer. This layer's vocabulary is:

Expression	**Meaning**
c_i: PROPOSE (op) , where $i \in \{1,..,n\}$	context c_i proposes operation op
NEWCONCEPT(d_j)	that is true iff concept d_j was created in the previous state
F: REQUEST(c_i : PROPOSE (op))	The framework requests context c_i to propose operation op

The top layer governs the general system because all action is initiated there.
Firstly, any local context can *initiate* the synchronization processes of the whole framework by activating the propose statement above assuming the framework mechanism is in waiting mode (and does not process another proposal then, e.g. is in “busy mode”). So this is the

formal sense in which the local contexts can exercise their semantic autonomy. After this statement is invoked, the framework mechanism invokes the corresponding procedural rules of the other layers. The top layer contains this rule:

$$\begin{array}{l} \text{F: DO (add\_ontorel(m, } c_j, d_j)) \Leftarrow \\ \text{NEWCONCEPT}(d_j) \Rightarrow \\ \text{F: REQUEST}(c_j : \text{PROPOSE(add\_mapping(m, } d_j, c_k))) \\ \text{where } k \in \{1,...,n\} \ \wedge \ k \neq j \end{array} \qquad \textbf{(9)}$$

This means that if an ontology relation *m* has actually been created within the ontology of a local context *j* and it connects a new concept d_j (to an existing concept c_j) then that local context is requested to "try" to generate proposals that would map this new concept to the other local contexts. It means that it has to ask the knowledge source to generate knowledge that fits that pattern, and sometimes that will actually result in this knowledge being generated. The rules are acting here as performative statements in a multi-agent system.

5.6 Computational Semantics of the Protocol

Here are the computational semantics of the definitions from §5.2. The type definitions use the class names that were defined at the beginning of §5.2.

5.6.1 A formalism for monitoring the execution of the protocol

Internally, the negotiation state ns contains different variables and their values. Also, it has a set of variables state_i (i=1..n) corresponding to all n occurrences of a command *com* $\in$ *FraCom* (as defined in §5.2) that actually occurs in a concrete protocol. It holds that $\text{state}_i=0$ for i=1...n before execution starts and this will change to $\text{state}_i=1$ once the corresponding command *com* actually has been executed. So if there are x instances of commands of the type *FraCom* in an actual protocol, then there will be a set of variables state_1, state_2 ... state_x corresponding to these instances. So these instances will be *executed* in some order. Therefore, this state variable says something about where a command instance is in its execution lifecycle. Concretely, then every instance of *com* is of the form $F : DO(op)$ where the *op* contains free variables. Initially, before execution starts, this holds:

ns$\vdash$ state$_i$=0 for every i=1..n (corresponding to all n occurrences of a command *com* $\in$ *FraCom*)

Secondly, we define these two special constraint predicates:

Execute_in(*com*, *ns*)
Executed_in(*com*, *ns*)

And their meaning is the following (where *i* is the unique identifier of *com*):

Execute_in(*com*, *ns*) is defined to mean that state$_i$ is set to 1 (*i* corresponds to the *i*:th occurrence of *com*).
Executed_in(*com*, *ns*) is defined as: ns$\vdash$ state$_i$=1

Executed_in(*com*, *ns*) returns true iff state$_i$=1, otherwise is returns false.
Informally, this will be used to keep track of if *com* has been executed or not, and the computational semantic below will set state$_i$ to 1, only if the *i:th com* has been executed.

We then define

contains(*Cexp*, *com*)

to mean that the command *Cexp* also executes the command *com*, i.e. includes that in its execution.

5.6.2 Defining different types of evaluation

In order to evaluate Boolean expressions into Boolean values we will use the evaluation arrow $\Rightarrow_B$.
The type definition for $\Rightarrow_B$ is
$Dec \times BExp \times NState \times OState \mapsto BVal$

The Boolean values T and F that will be used here, correspond true and false used in other chapters (and also to V[BExp]=1 and V[BExp]=0 used in chapter 3).

In order to evaluate commands and how they transform a state into another state we will need the evaluation arrow $\Rightarrow_C$.

The type definition for $\Rightarrow_C$ is

$Dec \times\ TopOrFCom \times NState \times OState \mapsto NState \times OState$

In order to evaluate a sphere expression and receive a set of spheres we need the evaluation arrow $\Rightarrow_S$.

The type definition for $\Rightarrow_S$ is

$Dec \times\ SpExp\ \times NState \times OState \mapsto SpFun$

In order to evaluate a context expression and receive a set of contexts we need the evaluation arrow $\Rightarrow$.

The type definition for $\Rightarrow$ is

$Dec \times\ ContExp\ \times NState \times OState \mapsto ContVal$

5.6.3 The computational semantics

The computation semantics for $\Rightarrow_B$ are the following:

$$\frac{}{D \vdash (T, ns, os) \Rightarrow_B T} \qquad \frac{}{D \vdash (F, ns, os) \Rightarrow_B F}$$

$$\frac{}{D \vdash (be \ \vee \ be', ns, os) \ \Rightarrow_B Or(bv, bv')}$$

$$\frac{\begin{array}{l} D \vdash (be, ns, os) \Rightarrow_B bv \\ D \vdash (be', ns, os) \Rightarrow_B bv' \end{array}}{D \vdash (be \ \wedge \ be', ns, os) \ \Rightarrow_B And(bv, bv')}$$

We defined the semantics for $\Rightarrow$:

$$\frac{}{D \vdash (contexp, ns, os) \Rightarrow conv}$$

The computation semantics for $\Rightarrow_C$ are the following:

$$\frac{\begin{array}{l} D \vdash (CT, ns = idle, os) \Rightarrow_C (ns', os) \\ D \vdash (CM, ns', os) \Rightarrow_C (ns'', os) \end{array}}{D \vdash CT \Rightarrow_C (ns'', os)}$$

and if contains(CT, *com*) that *com* $\in$ *FraCom* then Execute_in(*com*, ns'')

whenever $CT \Leftarrow CM$ occurs in D

We see that the pre-requisite for this execution is that the negotiations state is a special named state called *idle* because this rule is used for processing new proposals (and they can only be processed one at a time).

If contains(CF, *com*) that *com* $\in$ *FraCom* and Executed_in(*com*, *ns*)

$D \vdash (CF, ns, os) \Rightarrow_C (ns', os)$

$$\frac{D \vdash (CM, ns', os) \Rightarrow_C (ns'', os)}{D \vdash CF \Rightarrow_C (ns'', os)}$$

whenever $CF \Leftarrow CM$ occurs in D

$$\frac{\begin{array}{l} D \vdash (CM', ns, os) \Rightarrow_C (ns', os) \\ D \vdash (CM'', ns', os) \Rightarrow_C (ns'', os) \end{array}}{D \vdash (CM' \barwedge CM'', ns, os) \Rightarrow_C (ns'', os)}$$

$$\frac{\begin{array}{l} D \vdash (be, ns, os) \Rightarrow_B T \\ D \vdash (CFM', ns, os) \Rightarrow_C (ns', os) \end{array}}{D \vdash (be \Rightarrow CFM') \Rightarrow_C (ns', os)}$$

$$\frac{\begin{array}{l} D \vdash (be', ns, os) \Rightarrow_B T \\ D \vdash (CM', ns, os) \Rightarrow_C (ns', os) \end{array}}{D \vdash (be' \Rightarrow CM' \text{ or } be'' \Rightarrow CM'', ns, os) \Rightarrow_C (ns', os)}$$

$$\frac{\begin{array}{l} D \vdash (be', ns, os) \Rightarrow_B F \\ D \vdash (be'', ns, os) \Rightarrow_B T \\ D \vdash (CM'', ns, os) \Rightarrow_C (ns', os) \end{array}}{D \vdash (be' \Rightarrow CM' \text{ or } be'' \Rightarrow CM'', ns, os) \Rightarrow_C (ns', os)}$$

$$D \vdash (F : \mathit{DO}(op), ns, os) \Rightarrow_C (ns = idle, os')$$

os´ is a new ontology state where one of the four ontology operations have been performed on ontology in its state *os*, as defined in chapter 4. The negotiation state now changes to a state that has a special name, *idle¸* because it can again accept new proposals.

Chapter 6 A prototype implementation

In this chapter we will briefly describe a prototype implementation of the infrastructure mechanism. The prototype is implemented as a Java application in Java SE 5.0. The prototype system consists of these main parts (see Figure 20). We describe first the prototype of the infrastructure mechanism and its main parts:

6.1 Configuration & Sphere set-up

Before the infrastructure can process proposals it has first to be set up and configured. More precisely, these are the parameters that are decided at this stage:

- How many contexts and ontologies will there be?
  Currently, every context hosts one ontology.
- Which of the ontologies can grow and which of the ontologies can be connected by ontology mappings?
- Which spheres of consistency are defined, which ontology and mapping sets will they contain and what level of consistency will they maintain (i.e. the consistency parameter $p_c \in [0,1]$).

After these things are set up, then this configuration is (currently) kept constant during the execution (see Chapter 9 for future work about relaxing this assumption).

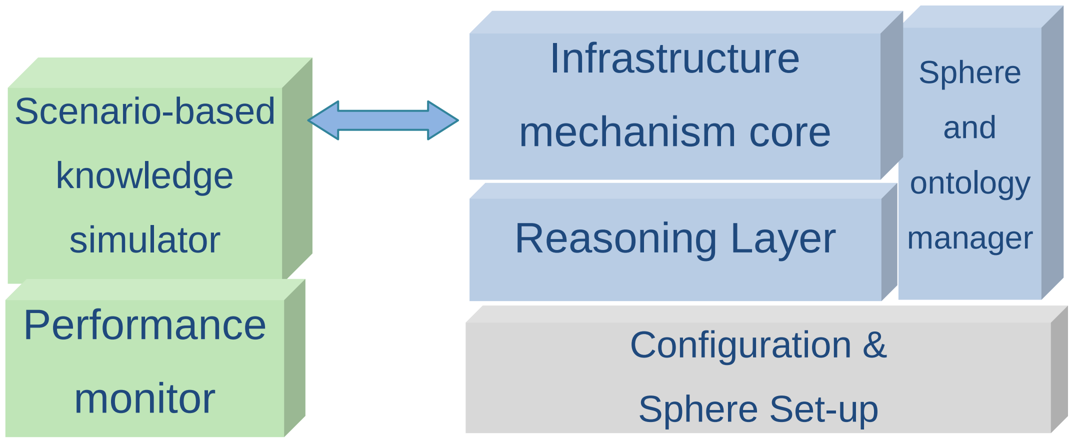

Scenario-based knowledge simulator
Performance monitor
Infrastructure mechanism core
Reasoning Layer
Sphere and ontology manager
Configuration & Sphere Set-up

Figure 20. A very abstract illustration of the infrastructure mechanism prototype (to the right) and its interaction with the scenario-based knowledge simulator (to the left).

6.2 Infrastructure mechanism core

This is the implementation of the rules described in Chapter 5, but with certain simplifications as described in §7.2. Incoming proposals are the events that drive the infrastructure mechanism and currently these proposals only come from the scenario-based knowledge simulator (see §6.5). In the future, it could have an open interface so that it could receive proposals that are "sent" to it from other applications (considering that this is the ultimate use of a knowledge infrastructure, as described in chapter 1). It is then processed according to the policy rules and it uses the sphere and ontology manager and the reasoning layer.

6.3 Sphere and ontology manager

This component manages all the created spheres and the ontologies of all the contexts.

6.4 The Reasoning Layer

The reasoning layer constitutes a big part of the implemented prototype. It implements the reasoning data structures and algorithms as they are described in Chapter 4. Often, it uses Hash-tables for storing big knowledge structures.

We will now mention the two modules that are develop only for the purpose of evaluating the infrastructure mechanism.

6.5 Scenario-based knowledge simulator

This module is not at a part of the infrastructure specification, but it is created for the scientific purpose of evaluating it. We first give a single integer as input and this describes which of the different scenarios we want to run (several different ones are described in Chapter 7). This means that every scenario contains a configuration that is used for generating connected ontologies. We will see that this knowledge simulation process uses a certain element of randomness in order to explore what happens in different situations where the semantic meaning is different.

Every individual scenario is defined by the following parameters:

- How many contexts and ontologies that exist.
- Which of the ontologies that are allowed to evolve and which of the pairs of ontologies that will develop ontology mappings (i.e. not all ontologies are connected).

- What the ratio is between small and big ontologies. We assume that ontologies are of two different sizes (e.g. that big ones are ten times larger than small ones). The knowledge simulator maintains this ratio simply by using the appropriate probabilities for which an ontology is proposed to grow by its contexts.
- A set of probabilities that define how probable it is that a certain context will randomly generate a proposal to evolve its ontology, or that it will propose to add a mapping between its ontology and another ontology.
- The parameters that will set up a set of spheres of consistency, i.e. the contexts they contain, the mapping sets and level of consistency.
- The amount of experiments that are done within a scenario. An experiment is defined as a session during which the whole system grows until it reaches it maximum size.
- The maximum total size to which the system grow (during every experiment).

6.6 Performance Monitor

The performance monitor measures the processing effort every time the infrastructure mechanism is processing a proposal. It measures these three parameters:

- The time it took the infrastructure mechanism to process the proposal
- The maximum memory used by the infrastructure mechanism to process the proposal
- The total size of the proof search trees that were created when the infrastructure mechanism processed the proposal

It also saves input parameters such as the size of the system when the proposal was received and sometimes other information, e.g. the degree of consistency.
Normally, for a given scenario at least 400 experiments are generated by the Scenario-based knowledge simulator, and the performance monitor collects the data generated during the course of every experiment. The results are then statistically analysed and examples of scenarios are presented in Chapter 7.

Chapter 7 . Evaluation Scenarios, Results and Analysis

7.1 Overview of experimental setup

We now present several scenarios where we evaluate our prototype system – more precisely its scalability in different circumstances. The purpose of this chapter is to investigate and measure how the computational effort (particularly time complexity) changes with the problem size (i.e. the total amount of mappings and ontology relationships within a sphere), for certain infrastructure configurations. We describe below the advantage choosing a general infrastructure configuration and then investigating the statistical average of 400 different choices of knowledge contents (particularly the types of mappings that connect ontologies), rather than investigating only a few ontologies. In this chapter we will evaluate what happens with the computational effort of an infrastructure mechanism when the spheres of consistency use proof-bounded reasoning of various degrees, when the size difference between small and large ontologies increases, when ontologies get wider and when the whole constellation of how ontologies are connected with each other and spheres of consistency are set up, changes.

The reason why we have not evaluated the system on "real" data is that such that data is very difficult to acquire and might not yet exist in reality – it would need to show how several inter-related ontologies and their mappings evolve over time. And this data log would also need to contain change proposals that were unsuccessful because they violate some assumed constraints – these would also need to be captured in detail. However, if an infrastructure mechanism that we propose in this thesis actually was fully developed and utilised, then it would be easier to acquire and analyse such data.

In order to avoid this problem, the scenarios are performed using simulated ontologies, but we have tried to make sure they have a structure that is analogous to that of real ontologies. For Example, when our simulations grow ontologies, they assume that the ontologies grow like recursive trees that have a certain branching factor. Also, in these scenarios (particularly 2 and 3) we present certain configurations of ontologies, mapping sets and spheres of consistency that are reminiscent of interesting situations that could occur in reality. More precisely, these situations could occur when several organizations that use ontology-based knowledge infrastructures would like to connect them and exchange formalised knowledge in a "safe" way with each other, i.e. making sure that semantic contradictions would not occur during the communication between a division in one organization and a division in another organization that both act as "interfaces" and exchange knowledge with each other. In a formal way, this will mean that these two interface ontologies are connected with mappings and that they are within at least one sphere of consistency that requires full consistency (in the default case). In these experiments we have also assumed that the "interface" ontologies are smaller than the "core" organizational ontologies, and we think this is a realistic assumption. We evaluate what happens when this ratio between "core" and "interface" ontologies increases.

The experimental evaluation presented here uses our prototype described in chapter 6 and it also utilizes the part of the prototype that generates problems with a certain configuration. These are the parameters that we vary between the different scenarios:

- The pairs of ontologies that have active connections between each other that will give rise to a growing amount of mappings that connect them.
- The particular spheres of consistency chosen, and their degree of consistency.
- The ratio between the size of "core" ontologies and the "interface" ontologies
- The recursive branching factor of all ontologies. The higher this factor is, the wider the ontologies are.

The first scenario investigates the effects of varying the amount bounded consistency, by varying the consistency parameter p_c defined in §4.3.1 from maintained perfect consistency to a state where all inconsistencies are allowed. The second scenario set investigates the benefits of full but pair-wise consistency compared to global consistency, in a situation where two organizations connect their ontologies with each other. Then we investigate what happens if the size difference between "core" and "interface" ontologies increases, and what happens if the ontologies that have to managed are wider. The third scenario investigates a three-organization topology where the interface ontologies of the three organizations have to be able to communicate with each other and therefore have to belong to the same sphere of consistency.

7.2 Specification of the experimental setting

We use a prototype implementation as specified and described in chapters 4, 5 and 6. However, the evaluation uses the following version of the rules, where the system does not try to remove subsets of existing ontologies when there is a contradiction, but instead it rejects the proposal. We decided to make this simplification, because we needed to focus on evaluation, and we will see several interesting phenomena nevertheless (a phase transition is one of them).

Case 1b. Proposing to add an ontology mapping.

$$\text{c}_\text{j}: \ \text{PROPOSE(add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{))} \Leftarrow$$

$$\left(\begin{array}{l} (\bigvee_{x \in spheres(c_j, c_k)} \text{F:C\_CONTRA(x, add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{)))} \Rightarrow \\ DO(\varepsilon) \end{array} \right) or \qquad \textbf{(10)}$$

$$\left(\begin{array}{l} (\bigvee_{x \in spheres(c_j, c_k)} \text{F: IS\_INFERABLE(x, add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{)))} \Rightarrow \\ \text{F: MCHOICE(\{c}_\text{j}\text{, c}_\text{k}\text{\}, "already\_known",} DO(\varepsilon),\ DO\text{(add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{)))} \end{array} \right) or$$

$$\left(\begin{array}{l} (\bigwedge_{x \in spheres(c_j, c_k)} \text{F: IS\_NEW(x, add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{)))} \Rightarrow \\ \text{F: MCHOICE(\{c}_\text{j}\text{, c}_\text{k}\text{\}, "new",} DO(\varepsilon),\ DO\text{(add\_mapping(m, c}_\text{j}\text{, c}_\text{k}\text{)))} \end{array} \right)$$

Case 3. Proposing to add an ontology relationship.

$$\text{c}_\text{j}: \ \text{PROPOSE(add\_ontorel(r, c}_\text{j}\text{))} \Leftarrow$$

$$\left(\begin{array}{l} (\bigvee_{x \in spheres(c_j)} \text{F:C\_CONTRA(x, add\_ontorel(r, c}_\text{j}\text{)))} \Rightarrow \\ DO(\varepsilon) \end{array} \right) or \qquad \textbf{(11)}$$

$$\left(\begin{array}{l} (\bigvee_{x \in spheres(c_j)} \text{F: IS\_INFERABLE(x, add\_ontorel(r, c}_\text{j}\text{)))} \Rightarrow \\ \text{F: MCHOICE(c}_\text{j}\text{, "already\_known",} \\ DO(\varepsilon),\ DO\text{(add\_ontorel(r, c}_\text{j}\text{)))} \\ \end{array} \right) or$$

$$\left(\begin{array}{l} (\bigwedge_{x \in spheres(c_j)} \text{F: IS\_NEW(x, add\_ontorel(r, } c_j\text{)))} \Rightarrow \\ \text{F: MCHOICE(}c_j\text{, "new",} \\ DO(\varepsilon),\ DO(\text{add\_ontorel(r, } c_j\text{)))} \end{array} \right)$$

The ontology operations that can occur in scenarios 1, 2 and 3 are the following ones:

- Add ontology mapping (between two concepts of different ontologies).
- Add ontology subsumption relation (between two concepts within an ontology).

As we have described in chapter 2, in a real application either human expert insight and knowledge or ontology or mapping induction would be used as the "source" that provides the proposals. Because it has been infeasible to use real data, we simulated the autonomy of the various contexts by having them make proposals that have a randomised semantic meaning instead of a meaning describing a real domain. In practice this means that a context proposes to add a mapping to the ontology of another context, then the mapping type is randomly chosen to be one of the five possible. Such a proposal is then processed by the infrastructure mechanism. In our scenarios we evolve the ontologies and mappings until the whole system has a certain size. The graphs will typically show the amount of proposals made along the x-axis and we should keep in mind that a fraction of these proposals are accepted (about 85-95%). For all scenarios, we generate a system of several networked ontologies (where the meaning of the mappings has been randomly generated) at least 400 times, and the graphs presented illustrate the average results and the statistical variation. We believe that this is an advantage of simulation, because we have then measured the behaviour of the system during hundreds of different possible knowledge settings, i.e. the semantic meaning of every such setting is different, and we can see both the average behaviour and the amount of variation.

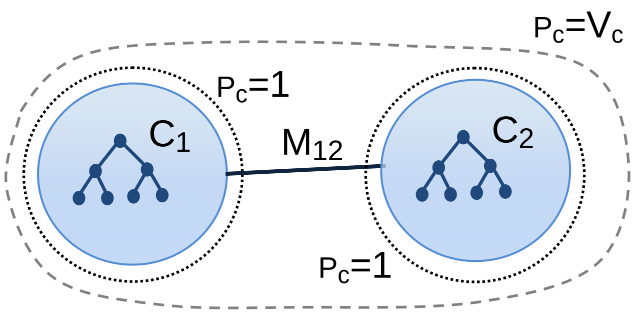


Figure 21. The contexts that have ontologies that 1) are internally consistent, 2) connected by a set of mappings M_{12} and where the whole system has a consistency of degree V_c.

7.3 Scenario 1. Investigating variable bounded consistency.

We now present an experimental evaluation where our implementation (as described in chapter 6) tests the core part of the framework (as described in §7.2 above). We assume two contexts c_1 and c_2 with one ontology each (see Figure 21). The ontologies of contexts c_1 and c_2 can evolve individually and mappings can evolve that connect these ontologies. We define spheres of consistency as illustrated in Figure 21, and we see that the largest sphere has a variable consistency we shall call v_c. So the three defined spheres are

Cons({c_1},{}, 1),
Cons({c_2},{}, 1),
Cons({c_1, c_2},{m_{12}}, v_c).

We now think of an application scenario where the contexts represent divisions within an organization, and all the "proposals" express organizational needs to adapt to a changing business environment. In this evaluation we assume that both contexts evolve ontologies having tree-like structures with a recursive branching factor 3, but it is randomized how quickly they grow. The expected size of the ontology of context c_1 and c_2 is the same, because the probability that any of them will grow is the same. These things can happen:

c_i : PROPOSE (Is ($C_{new,i}$, $C_{old,\,i}$))	A proposal to change(grow) the ontology *i* where the concept C_{new} is invented and added as a sub-concept of C_{old}
c_i : PROPOSE (add_mapping(m, c_i, c_k))	A proposal to add a mapping m from ontology *i* to *k*. One of the 5 mapping types is randomly chosen.

In other words, either one of the ontologies decides to grow or it proposes a random mapping from a recently created concept. During this simulation the system grows while obeying the rules and the sphere of consistency constraints. We have evaluated this implementation looking at the scalability and more precisely the incremental effort of the system to respond to a proposal and do the required reasoning. The total system size is 2000 in the experiment so e.g. d=2001 if v_c=1 but d=13 if v_c=1.994 (we solve equation 2 in section 4.3.1 for d). Our results are seen in Figure 22 and the time measured is the reasoning time that is needed when processing a proposal. The *slope* is defined as the constant *k* of the linear functions that are the best for the various line-like graphs in the top of Figure 22. Figure 24 and Figure 25 show the effort measured as proof search tree size and the memory use. These figures show the effort of the system to process proposals for adding the ontology mappings while maintaining all the above-mentioned constraints – so some proposals are rejected but others accepted. Every marked data point is the average for 100 consecutive proposals (x-values) for 420 different runs of the system (for every fixed v_c). We see that the difference between v_c=1 and v_c=1.994 is small, because the proofs of contradiction that occurs in an application domain having *this* structure have a depth that is small compared to the overall system size (but they could be very wide). One could therefore re-normalize parameter p_c (here v_c=p_c) depending on the application. However, in Figure 22 we observe a **phase transition** between two states with *different behaviour*: to the left of the transition the system has a stable linear time- and memory-complexity but to the right of it the system is close to being constant, in fact in that region it is approaching constant time- and memory-complexity.

7.3.1 Validation of the algorithm.

We have implemented a module that uses a slow brute-force method for measuring the smallest inconsistency in a set of ontologies and mappings – for any pair of variables it investigates if it can prove and disprove that any of the five mappings hold. We have generated 170 times a system of size 100 and validated that the measured inconsistency is never higher than the promised one (see below for more details about this validation).
In fact, often it is much lower (especially in a small system) because we have measured that the probability of a single proposal (in a system growing to the size of 2000) creating a contradiction is less than 0.6% for p_c=1 (it does decrease when p_c increases). But because of the potential *risk* that a contradiction could occur, reasoning about the existence of a contradiction still must be done.

The infrastructure relies constantly on the fact that has managed to maintain the defined form of consistency in its previous state, and only does the incremental work needed to make sure these constraints are still satisfied. In this validation, however, we did not make these assumptions at all but did an exhaustive search of all inconsistencies and listed them. We think the time complexity of this brute-force method was $O(n^3)$ and memory complexity $O(n)$ for measuring the size of the minimal inconsistency for this simple logic

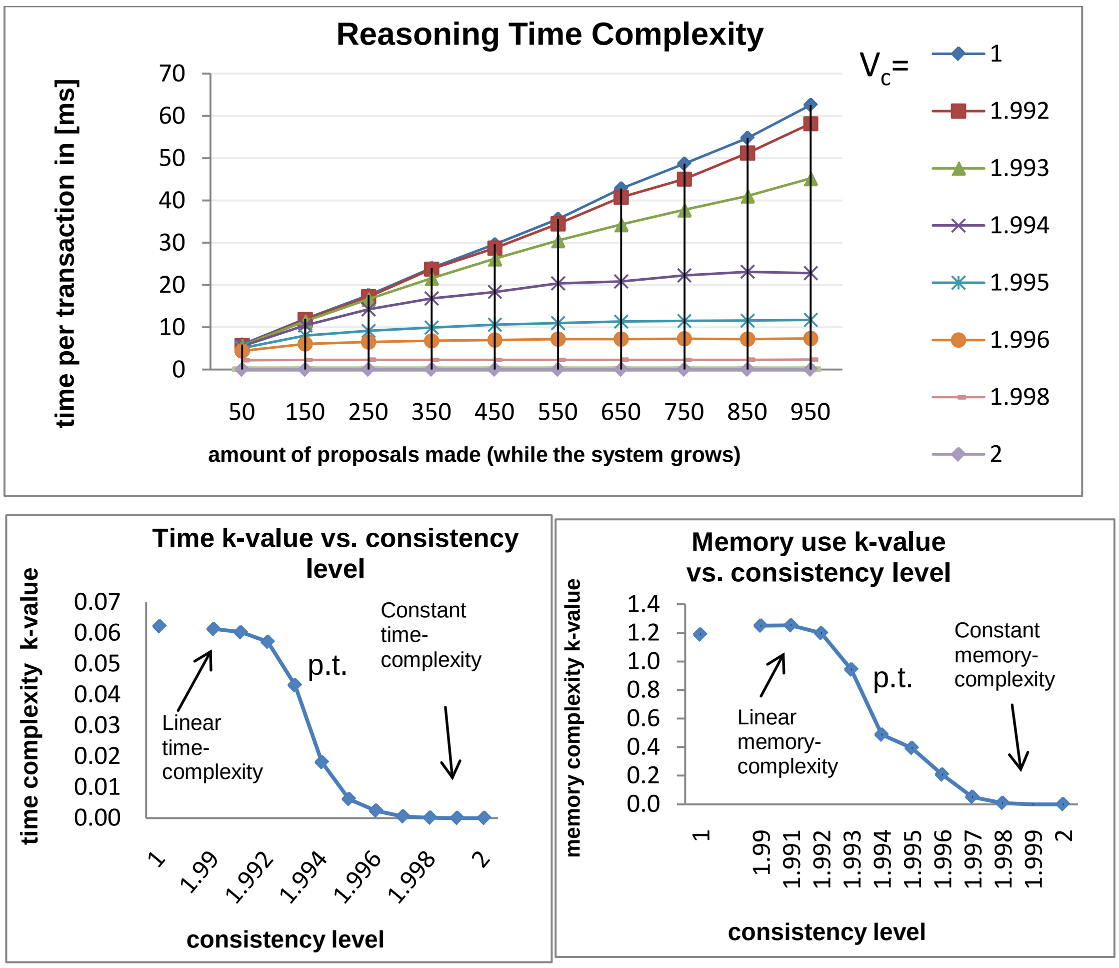


Figure 22. The figure on the top show the incremental time effort to process as single proposal for system where already x number of proposals have been processed and that has the degree of consistency V_c. The diagram on the bottom left side show the slope (i.e. all the *k:s* in that would be the best linear fit *y=kx+m* for the several line-like graphs above) of the top diagram lines, whereas the right diagram shows the analogous slopes for memory use.

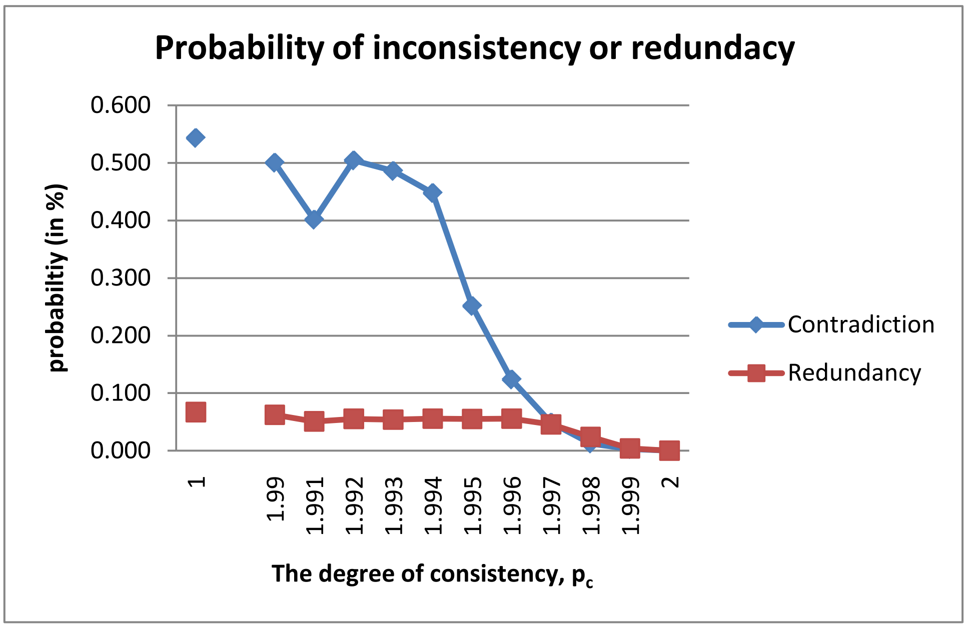


Figure 23. This chart shows statistics for scenario 1. It shows the probability that a certain proposal will be classified as contradictory, or that it will create redundancy vs. the degree of consistency. This means that almost all proposals in scenario 1 are classified as "new", i.e. they neither create contradiction nor redundancy. This graph is dependent on: the logic used, the reasoning rules and the semantics of the examples investigated. However, for a variety of such meta settings one can expect these lines to converge towards zero when p_c->2.

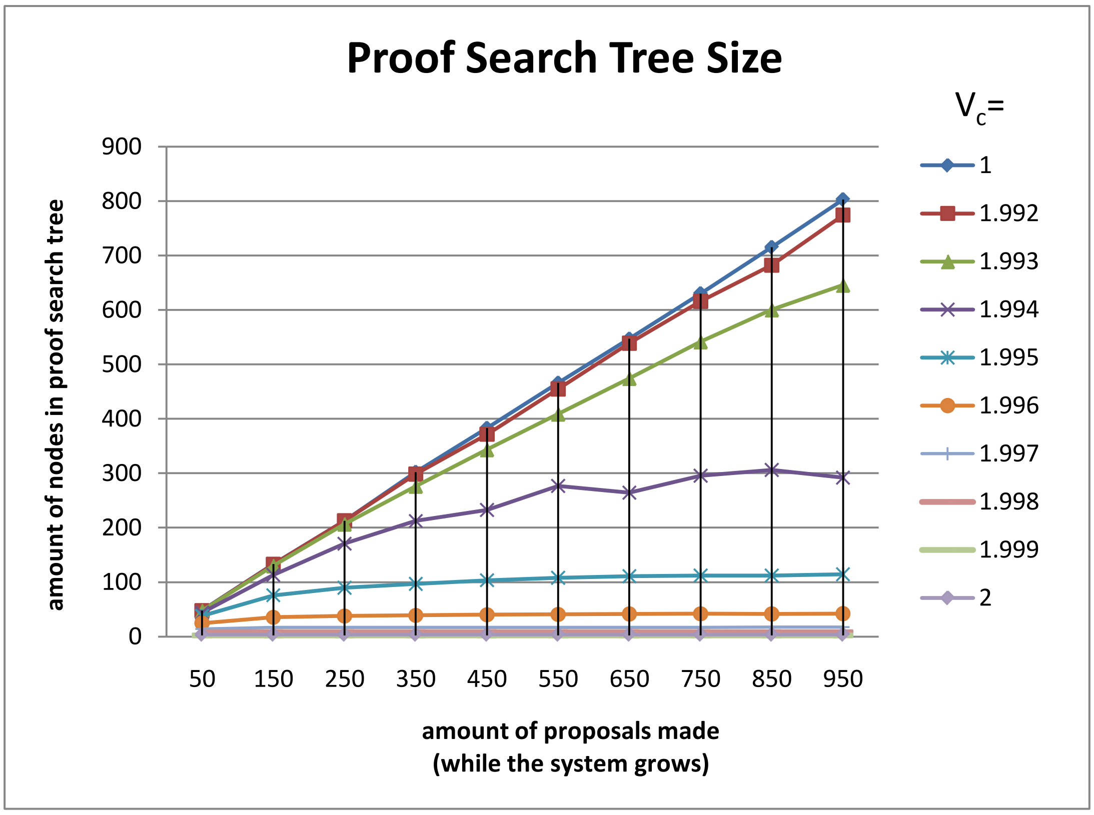


Figure 24. Given a certain degree consistency measure V_c and given a certain average proposal to process, this graph shows the average reasoning effort to process it, measured by the proof search tree size.

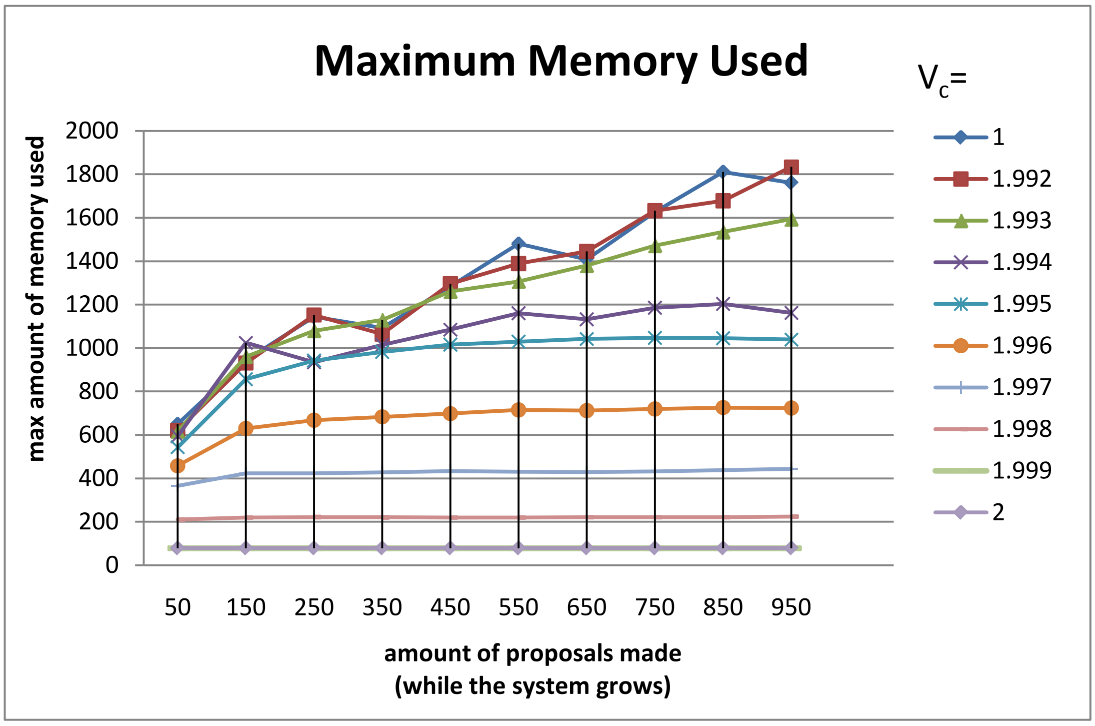


Figure 25. Given a certain degree of consistency V_c and given a certain proposal to process, this graphs shows the memory use (after the proposal has been processed the memory is released).

7.4 Scenario 2. Investigating the maintenance of pair-wise consistency using a two-organization topology.

7.4.1 Scenario 2.1. Pair-wise consistency using a two-organization topology.

The motivation of this scenario is that we have two organizations that have two ontologies each. Organization 1 has ontologies o_1 and o_2, whereas organization 2 has ontologies o_3 and o_4.

Ontologies o_2 and o_4 are seen as core ontologies whereas ontologies o_1 and o_3 are seen to be interface ontologies. The interface ontologies facilitate knowledge exchange between the two organizations that is delivered by the mappings between the two ontologies and the sphere of

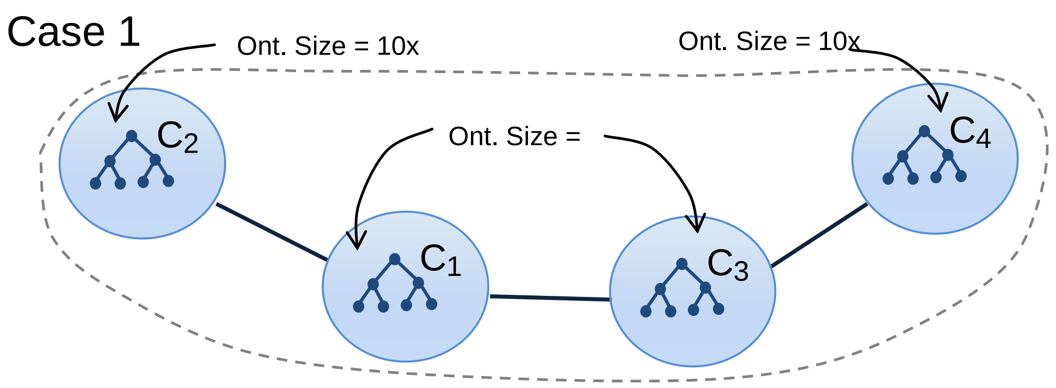


Figure 26. Global consistency is maintained between all the divisions of the two organizations.

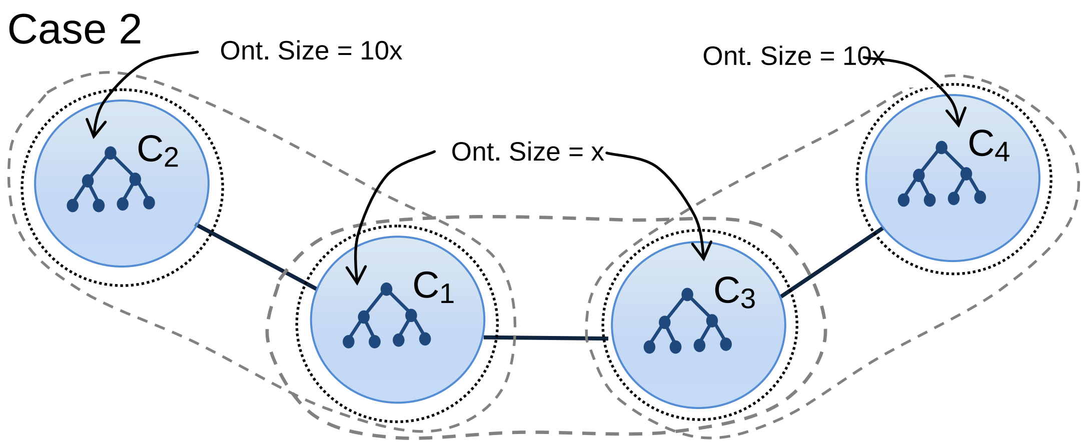


Figure 27. There is full consistency within every individual ontology, and within all ontologies and mappings of all organizations (i.e. C_1 and C_2 + mappings, and C_3 and C_4 + mappings) and between the two interface ontologies (C_1 and C_3 + mappings).

consistency that these two ontologies and their mapping set belong to (see Figure 26 and Figure 27). During the simulation contexts can either propose to evolve their ontologies, or to propose mappings to the ontologies of contexts to which they have an active relation (this is exemplified in Scenario 1). We now describe this scenario formally.

There are four contexts: c_1, c_2, c_3 and c_4 and they all have an ontology each.

The set of active relations between the contexts is:

$\{(c_1, c_2), (c_2, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_4), (c_4, c_3)\}$

In Case 1 these are the spheres of consistency:

$\text{Cons}(\{c_1, c_2, c_3, c_4\},\{m_{12}, m_{13}, m_{34}\}, 1)$.

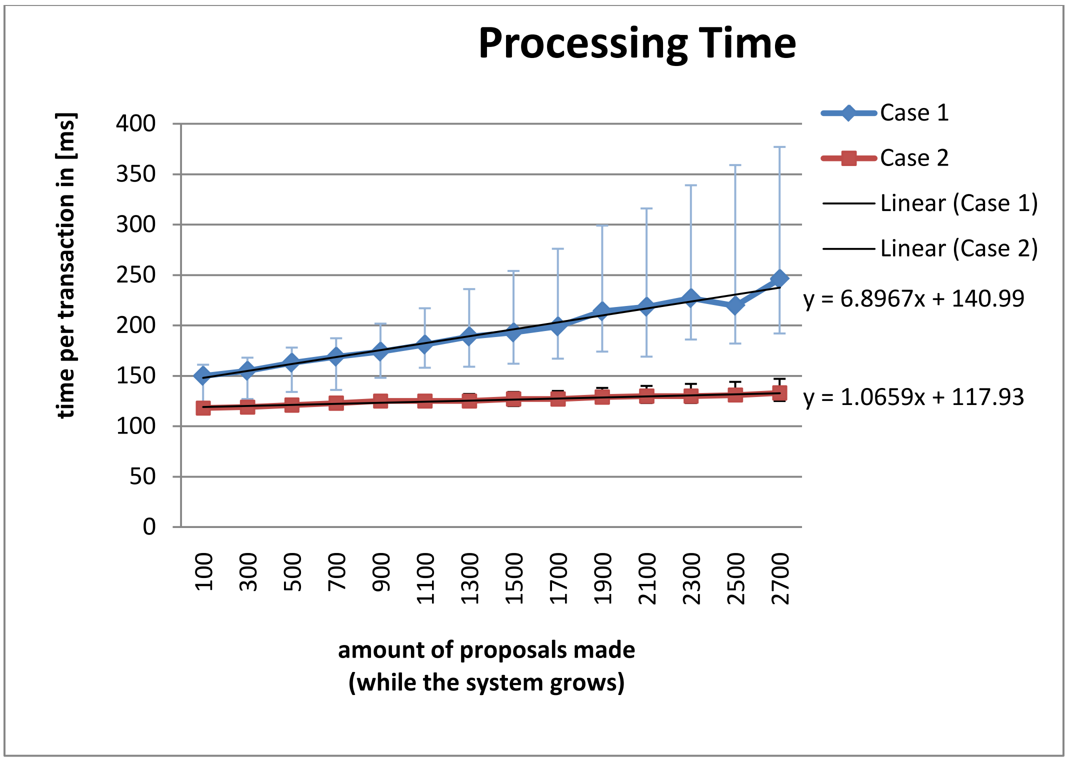


Figure 28. The time for the system to process a proposal in case 1 vs. case 2, compared with how many proposals that already have been processed (which is related to total system size).

In Case 2 these are the spheres of consistency:

$\text{Cons}(\{c_1\},\{\}, 1)$,

$\text{Cons}(\{c_2\},\{\}, 1)$,

$\text{Cons}(\{c_3\},\{\}, 1)$,

$\text{Cons}(\{c_4\},\{\}, 1)$,

$\text{Cons}(\{c_1, c_2\},\{m_{12}\}, 1)$.

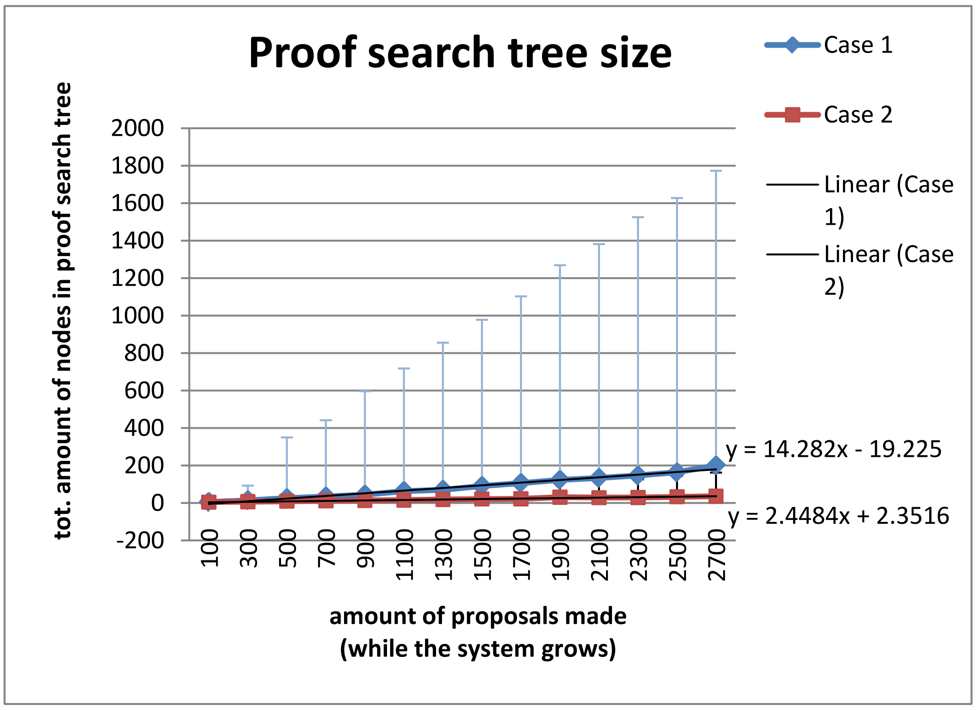


Figure 29. The y-axis here represents the total size of all proof search trees generated (i.e. one tree per sphere investigated) when a single proposal is processed. The x-axis is the same as before. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1.

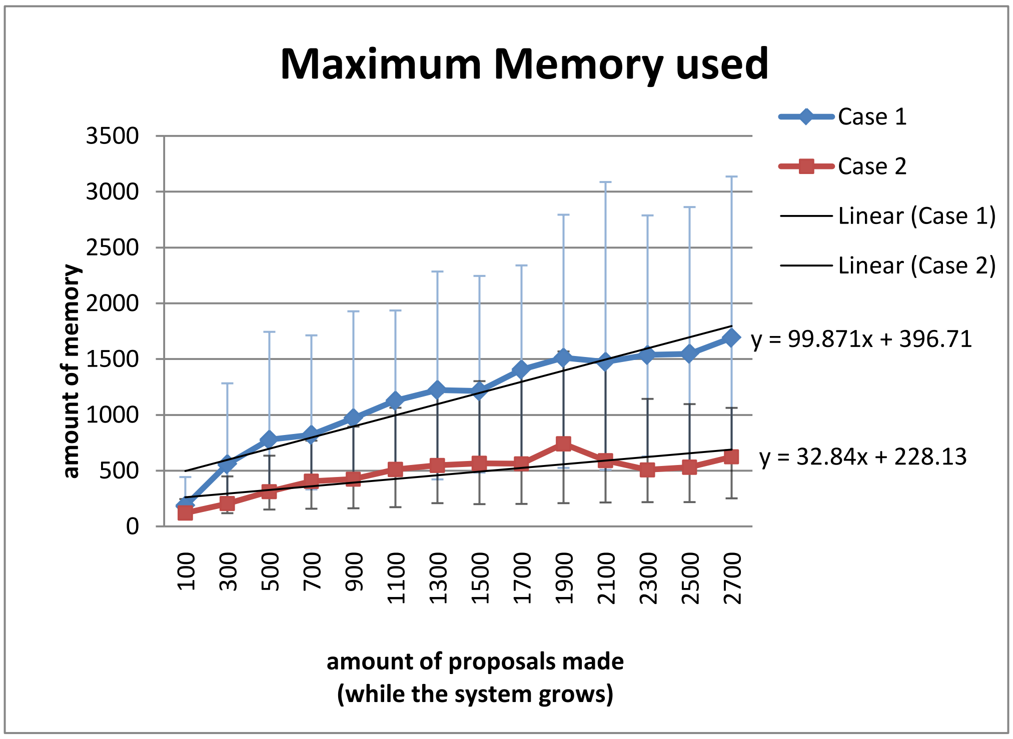


Figure 30. The y-axis shows the maximum memory used during the processing of a single proposal. If several spheres of consistency are investigated, then the one using most memory will be shown. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1.

Cons($\{c_1, c_3\}$,$\{m_{13}\}$, 1).
Cons($\{c_3, c_4\}$,$\{m_{34}\}$, 1).

The evolution of ontologies and mappings is based on proposals that are generated randomly with certain probabilities. However, these probabilities are chosen in such way that if the statistically expected size of ontology 1 and 3 is x, then the statistically expected size of ontology 2 and 4 is 10x. All the four ontologies (o_1, o_2, o_3 and o_4) have the branching factor 3.

We have measured the scalability of the system and investigated the computational effort to process an evolution proposal (see Figure 28, Figure 29 and Figure 30). This effort is measured in a three-fold way: the time it takes to process the proposal, the total sum of the sizes of proof search trees that are built during the process and the maximum memory used to process the proposal. As regards the second and third measure, we should consider that when the reasoning looks into several spheres of consistency, then it does reasoning separately in all of them so the effort is split into chunks. In the case of global consistency, one big proof search tree is built and memory is allocated for that single larger effort.

It makes sense to measure the incremental effort of the infrastructure mechanism to process a proposal, because then it remains in a "stable" state where all the consistency constraints are satisfied and it waits for the next proposal. The reason for doing this scalability investigation is to see if the infrastructure mechanism could mange of big set of ontologies when it is configured in a certain way. We learn from Figure 28 that the incremental efforts grows linearly in both cases, but that the processing time is more than 6 times (6.4 exactly) smaller in case 2 compared with case 1. The proposals included in these statistics are when contexts 1 and 3 (that host interface ontologies) make proposals to each other to add mappings between their ontologies.

The x-axis shows the amount of proposals accepted or rejected, and for every run of the generated scenario (the average of 400 runs is presented in the graphs) the simulations stops when the total system size is 2000. That end point will be at the x-axis between the proposal value 2000 and 2800 and varies between every run.

7.4.2 Scenario 2.2. Pair-wise consistency with an increased size difference between ontologies.

In this scenario we again have two organizations having two ontologies each (see Figure 31 and Figure 32). We can envision that organization 1 has ontologies 1 and 2 and the mappings between them, whereas organization 2 has ontologies 3 and 4. As before, contexts c_2 and c_4 host the core ontologies, whereas c_1 and c_3 have the interface ontologies. During the simulation contexts can either propose to evolve their ontologies, or to propose mappings to the ontologies of contexts to which they have an active relation (this was exemplified in Scenario 1).

The formal model is as follows.
There are four contexts: c_1, c_2, c_3 and c_4 and they all have an ontology each.
The set of active relations between the contexts is:
$\{(c_1, c_2), (c_2, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_4), (c_4, c_3)\}$

In Case 1b these are these spheres of consistency:
$Cons(\{c_1, c_2, c_3, c_4\},\{m_{12}, m_{13}, m_{34}\}, 1)$.

In Case 2b these are these spheres of consistency:

$Cons(\{c_1\},\{\}, 1)$,
$Cons(\{c_2\},\{\}, 1)$, $Cons(\{c_3\},\{\}, 1)$,
$Cons(\{c_4\},\{\}, 1)$,
$Cons(\{c_1, c_2\},\{m_{12}\}, 1)$.
$Cons(\{c_1, c_3\},\{m_{13}\}, 1)$.
$Cons(\{c_3, c_4\},\{m_{34}\}, 1)$.

The evolution of ontologies and mappings is based on proposals that are generated randomly with certain probabilities. However, these probabilities are chosen in such way that if the statistically expected size of ontology 1 and 3 is x, then the statistically expected size of ontology 2 and 4 is 20x. This means that compared to scenario 2.1 the size ratio between the core and interface ontologies is now larger. The reason why this is interesting is that it gives us a sensitivity analysis where we can measure if the increased size difference between the small and large ontologies benefits more from more confined spheres of consistency

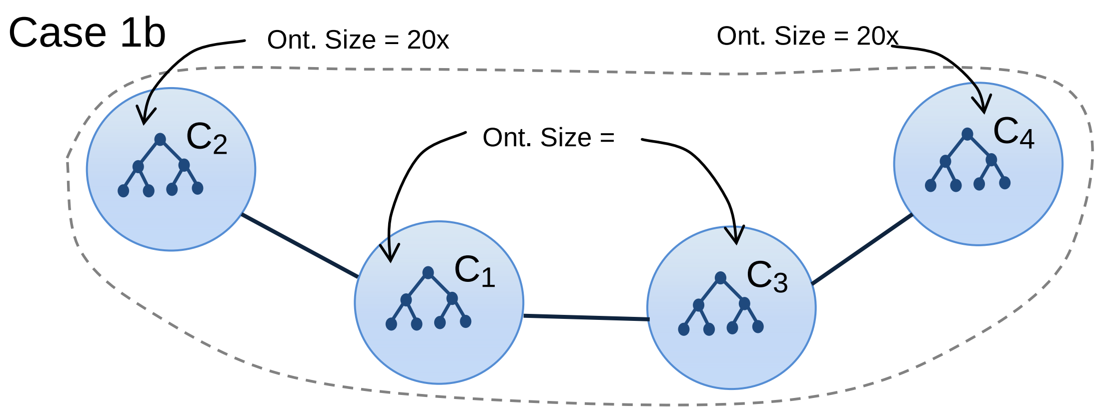


Figure 31. Global consistency is maintained between all the divisions of the two organizations.

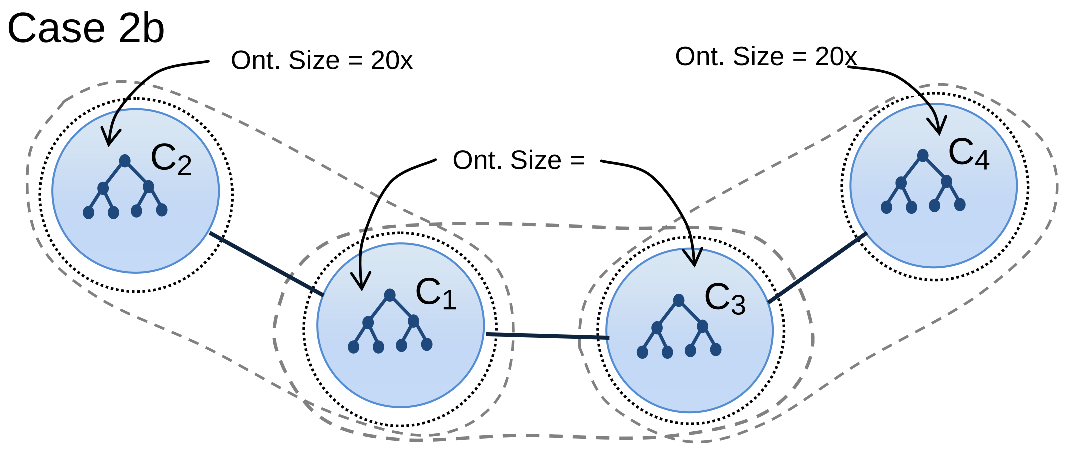


Figure 32. There is full consistency within every individual ontology, and within all ontologies and mappings of all organizations (i.e. C_1 and C_2 + mappings, and C_3 and C_4 + mappings) and between the two interface ontologies (C_1 and C_3 + mappings).

compared to the case where the size difference is smaller. All the four ontologies (o_1, o_2, o_3 and o_4) have the branching factor 3. The graphs evaluate the computational effort of the infrastructure mechanism to process proposals that context 1 makes that would map its ontology to the ontology of context 3. We see the results in Figure 33, Figure 34 and Figure 35.

We learn from Figure 33 that the effort to process a proposal grows linearly in both case 1b and 2b, but that the growth of these linear functions is 9 times higher for case 1b, i.e. when global consistency is maintained. If we compare with scenario 2.1 and its case 1, where this ratio was around 6, we learn that if the size difference between the large core ontologies and smaller interface ontologies grows, the system will benefit even more from using pair-wise consistency. We also learn from Figure 33 that case 1b has a much higher variability than case 2b, so using the infrastructure for managing pair-wise consistency makes the system behaviour more predictable. Figure 34 illustrates these ideas as well, but it shows the computation effort in the form of the total size of proof search trees, instead of time. Finally, Figure 35 illustrates the memory that is allocated for the case 1b vs. 2b. These could be hypothesized to grow linearly (or at least to be bounded b y a linear function) while this linear function grows twice as fast for case 1b compared with case 2b, i.e. this should converge towards memory use in case 1b being double the memory use in case 2b. So that is yet an advantage of managed pair-wise consistency. However, the variability of the memory use is very high. We believe this is due to the fact that Java has been used and it has a virtual machine that makes decision about memory allocation and garbage collection, and also because the implementation always creates hash-tables with the ontological knowledge, for fast access.

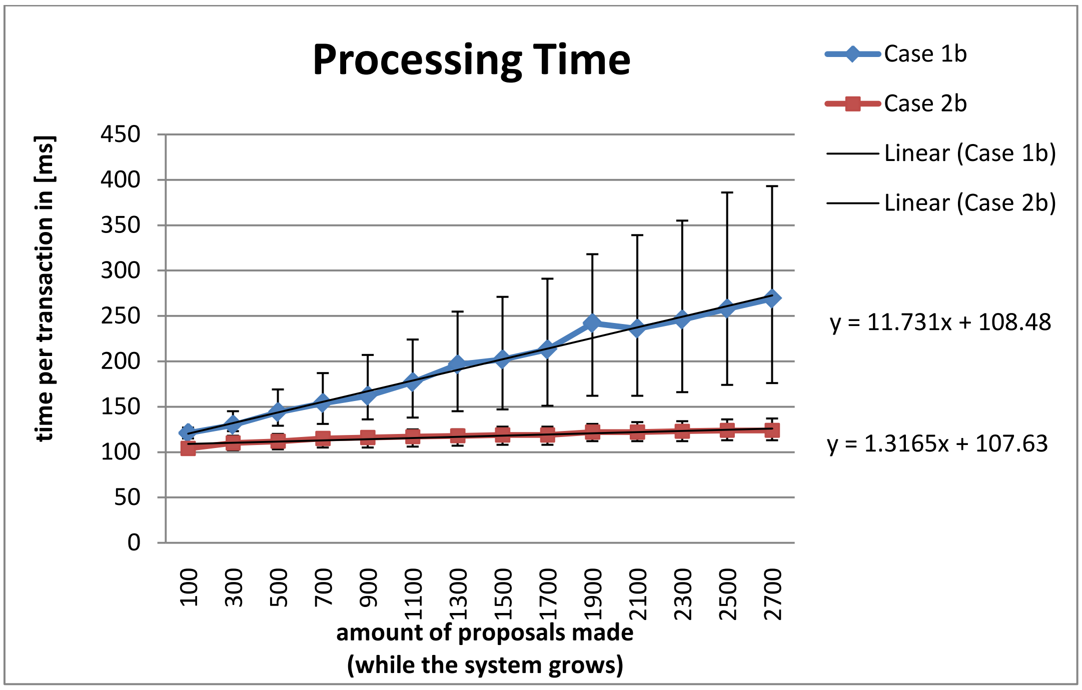


Figure 33. The time for the system to process a proposal in case 1b vs. case 2b, compared with how many proposals that already have been processed (which is related to total system size).

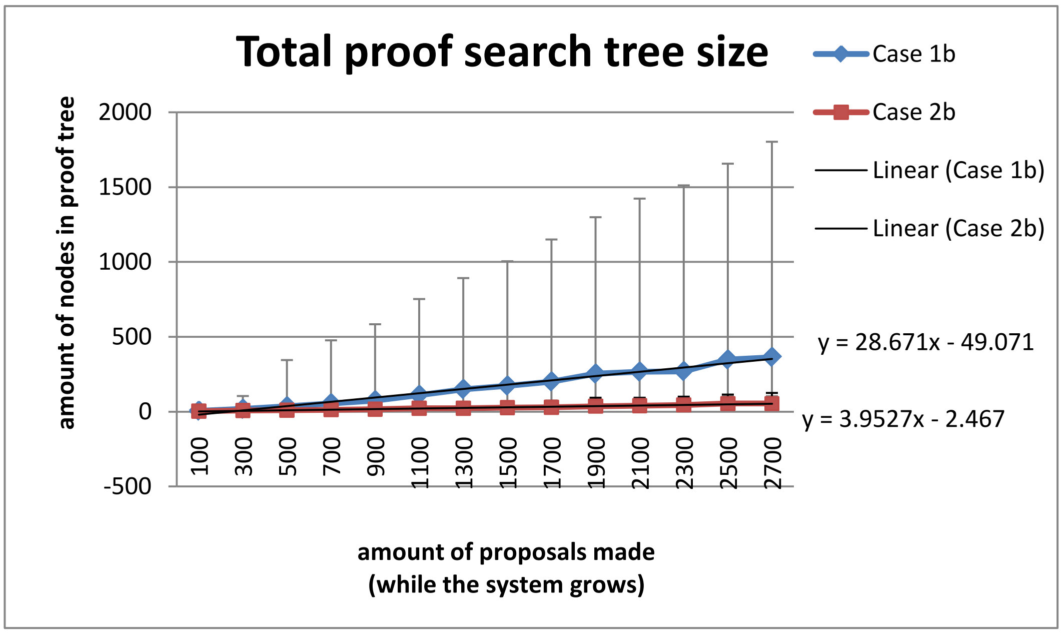


Figure 34. The y-axis here represents the total size of all proof search trees generated (i.e. one tree per sphere investigated) when a single proposal is processed. The x-axis is the same as before. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1b.

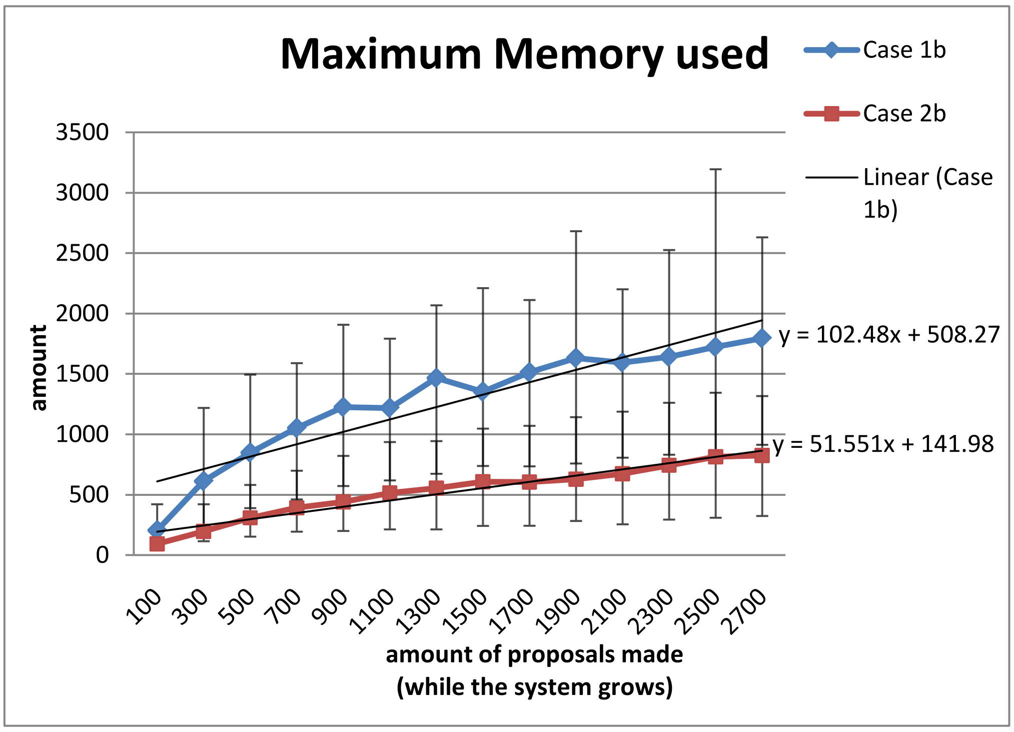


Figure 35. The y-axis shows the maximum memory used during the processing of a single proposal. If several spheres of consistency are investigated, then the one using most memory will influence the result. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1.

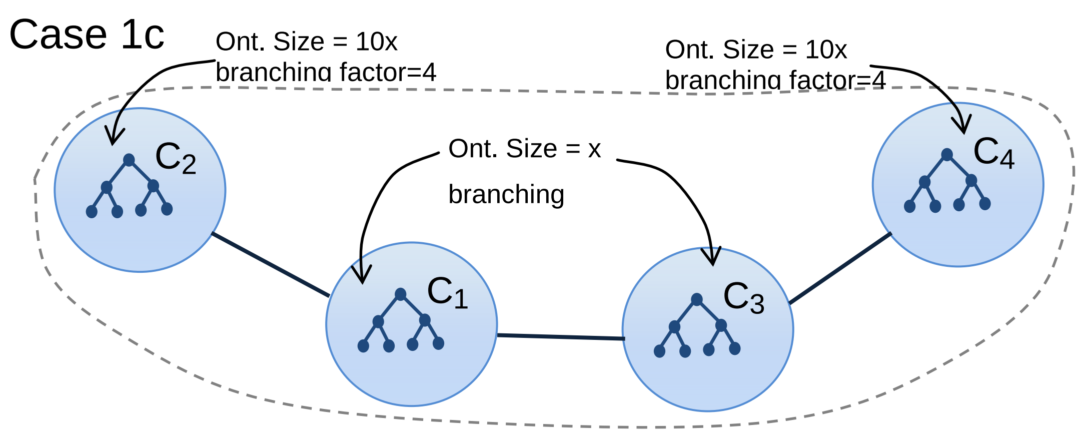


Figure 37. Global consistency is maintained between all the (ontologies of) the divisions of the two organizations.

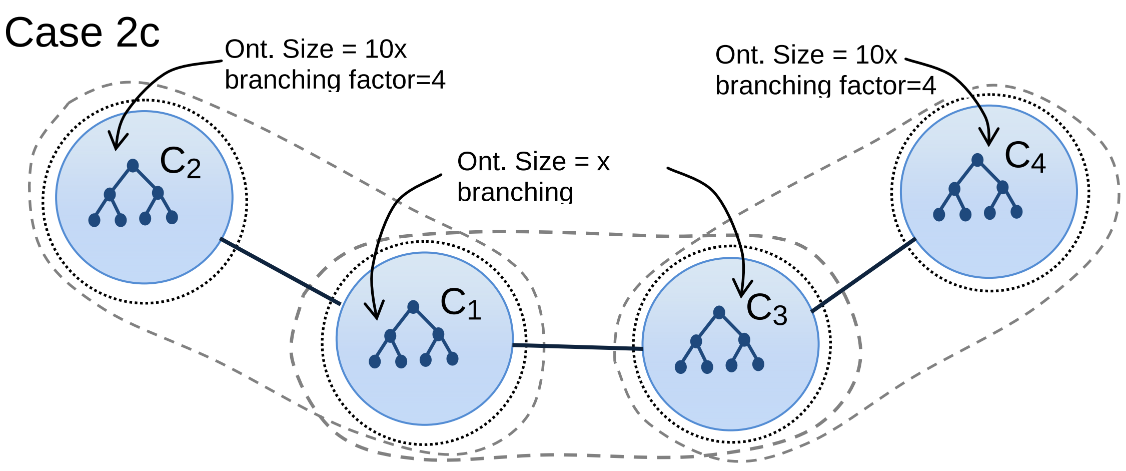


Figure 36. There is full consistency within every individual ontology, and within all ontologies and mappings of all organizations (i.e. C1 and C2 + mappings, and C3 and C4 + mappings) and between the two interface ontologies (C1 and C3 + mappings).

7.4.3 Scenario 2.3. Pair-wise consistency using wider ontologies.

In this scenario we again have two organizations having two ontologies each (Figure 37 and Figure 36). As before, contexts c_2 and c_4 have the core ontologies, whereas c_1 and c_3 have the interface ontologies. During the simulation contexts can either propose to evolve their ontologies, or to propose mappings to the ontologies of contexts to which they have an active relation (this exemplified in Scenario 1).

There are four contexts: c_1, c_2, c_3 and c_4 and they all have an ontology each.

The set of active relations between the contexts is:

$\{(c_1, c_2), (c_2, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_4), (c_4, c_3)\}$

In Case 1 these are the spheres of consistency:

Cons({c_1, c_2, c_3, c_4},{m_{12}, m_{13}, m_{34}}, 1).

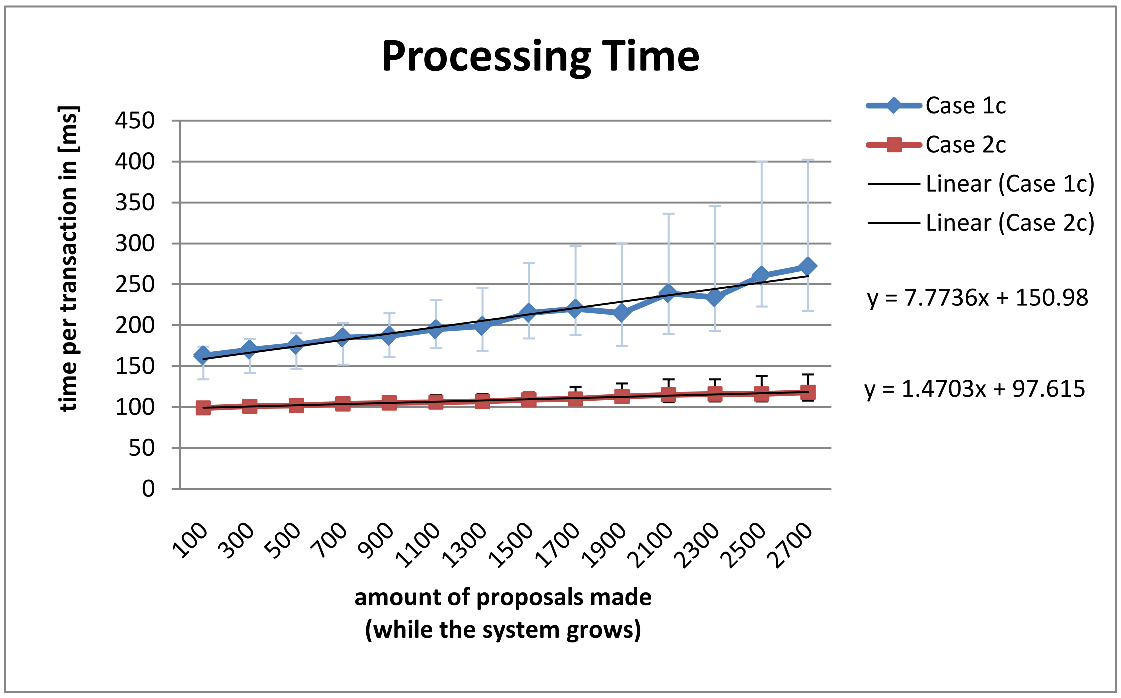


Figure 38. The time for the system to process a proposal in case 1c vs. case 2c, compared with how many proposals that already have been processed (which is related to total system size).

In Case 2 these are the spheres of consistency:

Cons({c_1},{}, 1),

Cons({c_2},{}, 1),

Cons({c_3},{}, 1),

Cons({c_4},{}, 1),

Cons({c_1, c_2},{m_{12}}, 1).

Cons({c_1, c_3},{m_{13}}, 1).

Cons({c_3, c_4},{m_{34}}, 1).

The evolution of ontologies and mappings is based on proposals that are generated randomly with certain probabilities. However, these probabilities are chosen in such way that if the statistically expected size of ontology 1 and 3 is x, then the statistically expected size of ontology 2 and 4 is 10x.

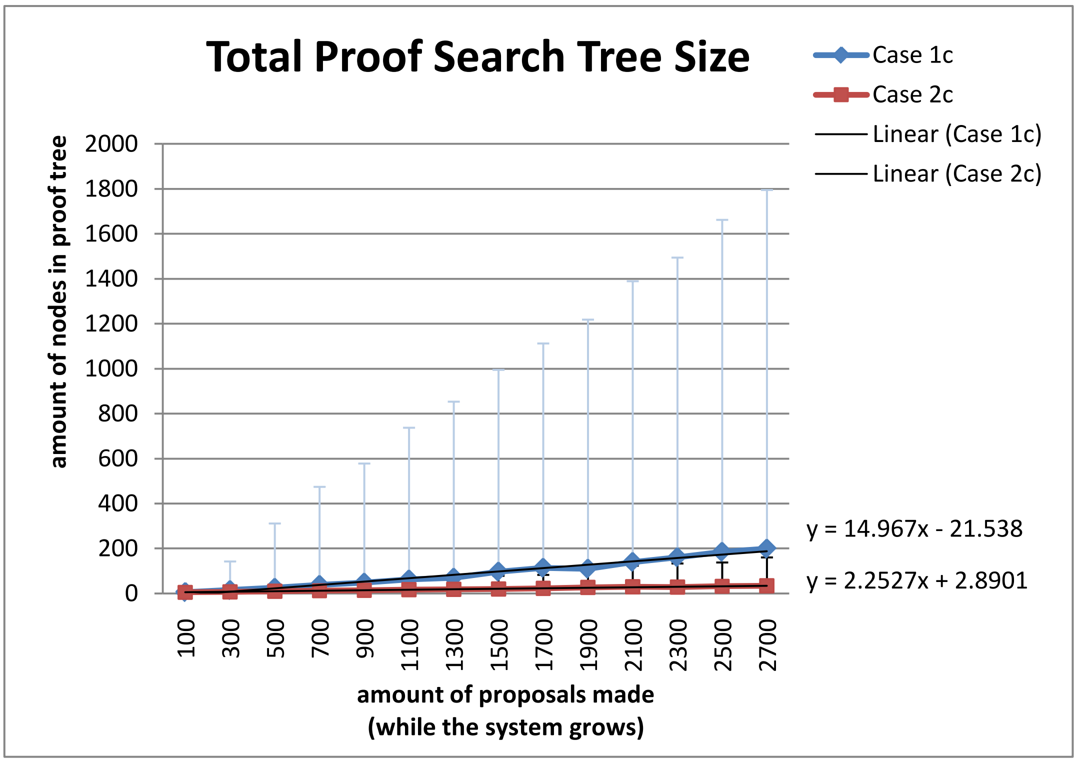


Figure 39. The y-axis here represents the total size of all proof search trees generated (i.e. one tree per sphere investigated) when a single proposal is processed. The x-axis is the same as before. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1c.

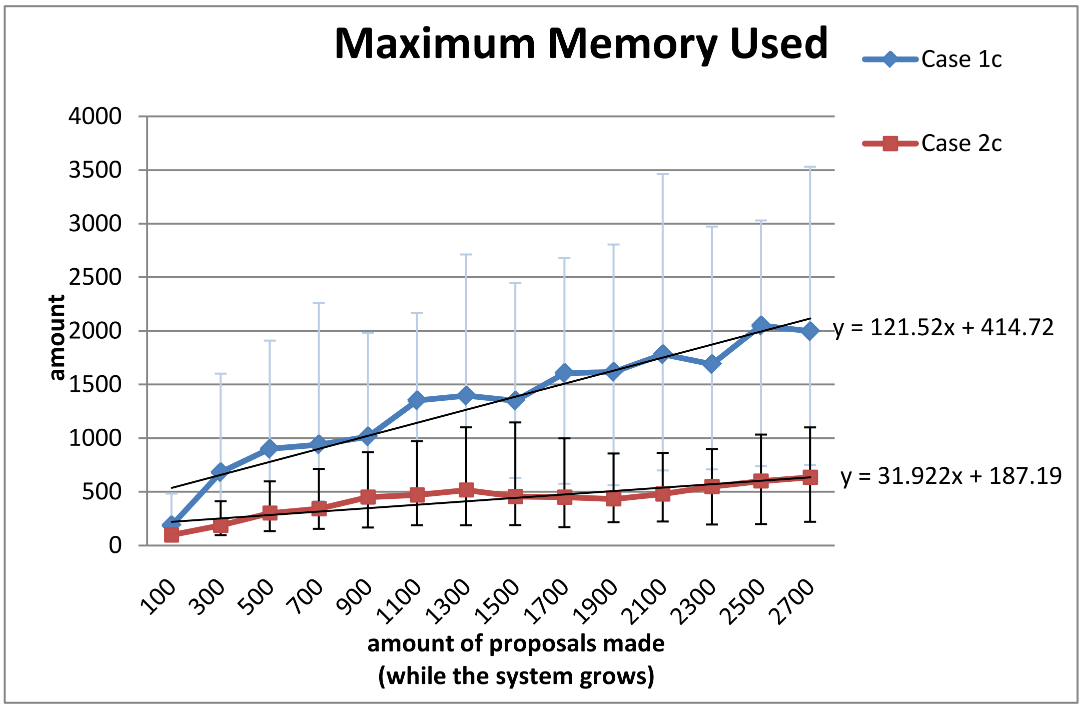


Figure 40. The y-axis shows the maximum memory used during the processing of a single proposal. If several spheres of consistency are investigated, then the one using most memory will influence the result. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1c.

All the four ontologies (o_1, o_2, o_3 and o_4) have the branching factor 4. When the ontologies are wider this could affect the reasoning effort because it implies that ontology viewed as a graph has higher connectivity compared with scenarios 2.1 and 2.2.

We learn from Figure 38 that the incremental effort to process a proposal grows linearly in both case 1c and 2c, but that the growth of these linear functions is 5.2 times higher for case 1c, i.e. when global consistency is maintained. If we compare with scenario 2.1 and its case 1 and case 2, we recall that it was 6.4, i.e. it was greater. So it seems that that the higher connectivity (caused by wider ontologies) reduced the difference somewhat. However, the ratio that measures the effort considering total proof search size in Figure 39 is 6.6 (vs. 5.9 in scenario 2.1).

The measured time effort is still more important, so it seems that wider ontologies reduce the benefit somewhat of pair-wise consistency, but more experiments are needed to say something more firm about this. From a network science point of view (see chapter 2), wider ontologies implies higher connectivity of the "knowledge network" that is created, i.e. the network of connected ontologies.

Also, as before, we see that if global consistency is used the variability of the system performance is much greater compared to automatically managed pair-wise consistency.

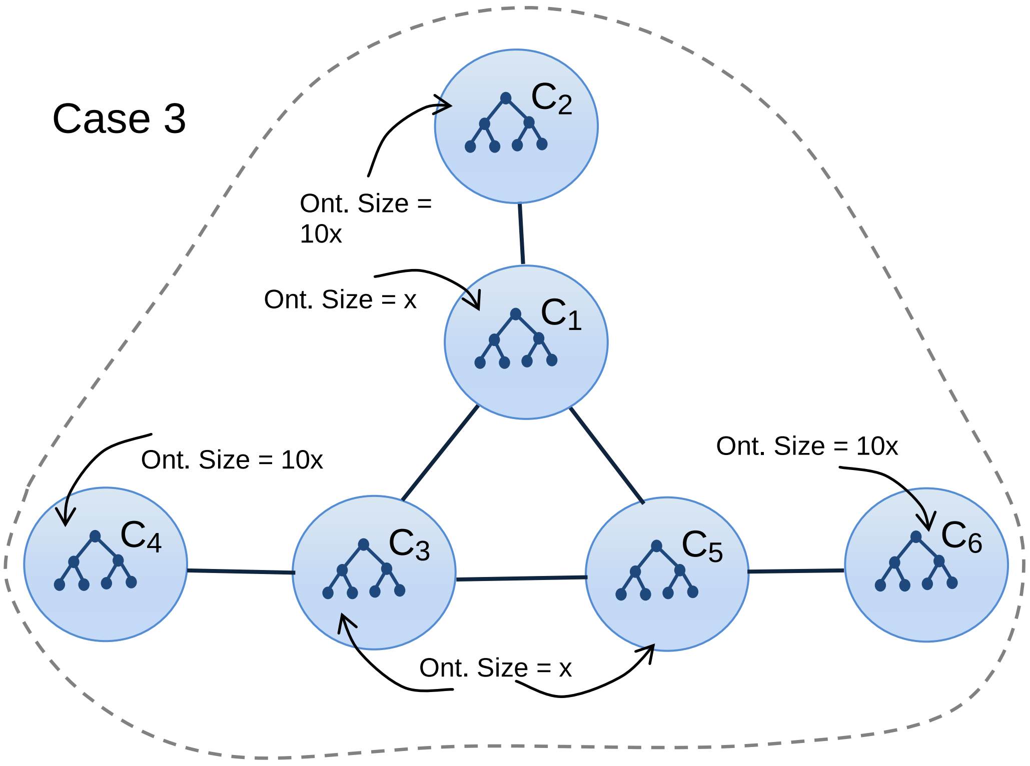


Figure 41. Global consistency is maintained between the ontologies of all the divisions of the three organizations.

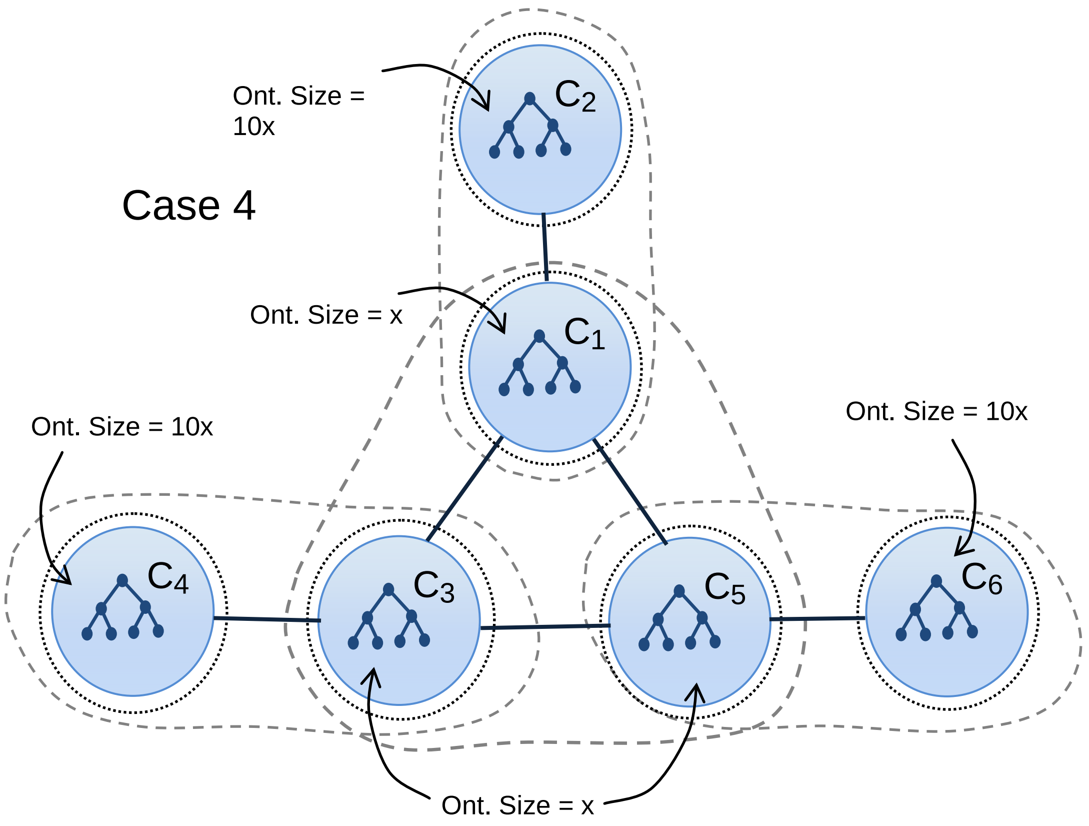


Figure 42. There is full consistency within every individual ontology, and within all ontologies and mappings of all organizations (i.e. C_1 and C_2 + mappings, and C_3 and C_4 + mappings, C_5 and C_6 + mappings) and between the three interface ontologies (C_1 + C_3 + C_5 + mappings).

7.4.4 Scenario 3. Investigating the maintenance of pair-wise consistency using a three-organization topology.

The third scenario will investigate a three-organization topology, where every organization is considered to have two divisions, that will be represented by two contexts having an ontology each, i.e. one per division (see Figure 41 and Figure 42). For every such pair of ontologies, one ontology is considered to be a core ontology whereas the other one is a interface ontology, i.e. it is used to exchange knowledge with other interface ontologies. Ontologies o_2, o_4 and o_6 are considered to be core ontologies in this scenario, whereas ontologies o_1, o_3 and o_5 are considered to be interface ontologies because a sphere of consistency maintains full consistency between them (for the purpose of their communication that could utilize these ontologies and the mappings between them). This scenario is different from the previous ones in a particular way: the sphere of consistency connecting interface ontologies now contains three ontologies that are connected by mappings. In contrast to before, this creates more constraints due to the "loop" created by the mappings. E.g. if the context of ontology 1 proposes to add a mapping to ontology 3, then the reasoning in that sphere will investigate if the proposal is consistent with the existing mappings between ontology 1 and ontology 3, but it will also investigate if any contradictions are created by going via the ontology of context 5 and the mappings that lead via ontology 5. From an application point of view, the ontologies 1, 3 and 5 and the mappings between them are representing a form of consensus that could be used by semantic applications (e.g. negotiations about products and prices) where these tree ontologies would be involved (and therefore, also the organizations that they are a part of).

This configuration can be described formally in this way.
There are six contexts: c_1, c_2, c_3, c_4, c_5 and c_6 and they all have an ontology each.
The set of active relations between the contexts is:
$\{(c_1, c_2), (c_2, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_4), (c_4, c_3), (c_5, c_6), (c_6, c_5), (c_3, c_5), (c_5, c_3), (c_1, c_5), (c_5, c_1)\}$

In Case 1 these are the spheres of consistency:
$\text{Cons}(\{c_1, c_2, c_3, c_4, c_5, c_6\},\{m_{12}, m_{13}, m_{34}\}, 1)$.

In Case 2 these are the spheres of consistency:
$\text{Cons}(\{c_1\},\{\}, 1)$,
$\text{Cons}(\{c_2\},\{\}, 1)$,
$\text{Cons}(\{c_3\},\{\}, 1)$,
$\text{Cons}(\{c_4\},\{\}, 1)$,
$\text{Cons}(\{c_5\},\{\}, 1)$,
$\text{Cons}(\{c_6\},\{\}, 1)$,
$\text{Cons}(\{c_1, c_2\},\{m_{12}\}, 1)$.
$\text{Cons}(\{c_3, c_4\},\{m_{34}\}, 1)$.
$\text{Cons}(\{c_5, c_6\},\{m_{56}\}, 1)$.
$\text{Cons}(\{c_1, c_3, c_5\},\{m_{13}, m_{35}, m_{15}\}, 1)$.

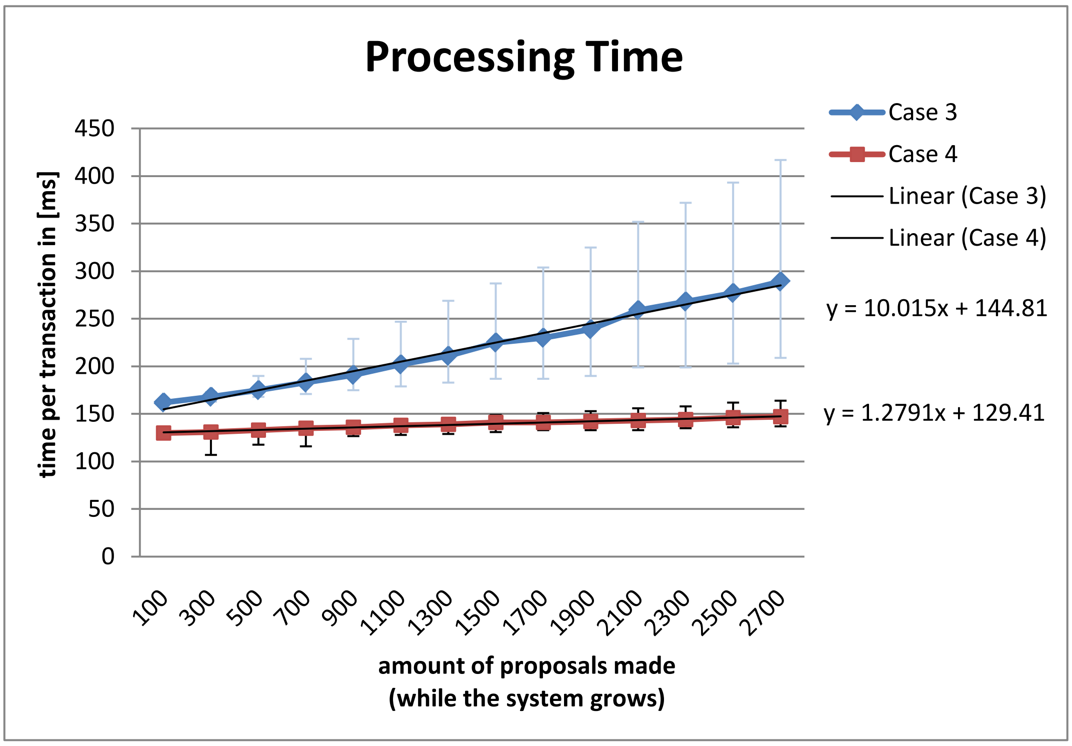


Figure 43. The time for the system to process a proposal in case 3 vs. case 4, compared with how many proposals that already have been processed (which is related to total system size). The error bars show the 25% and 75% percentiles for the two cases.

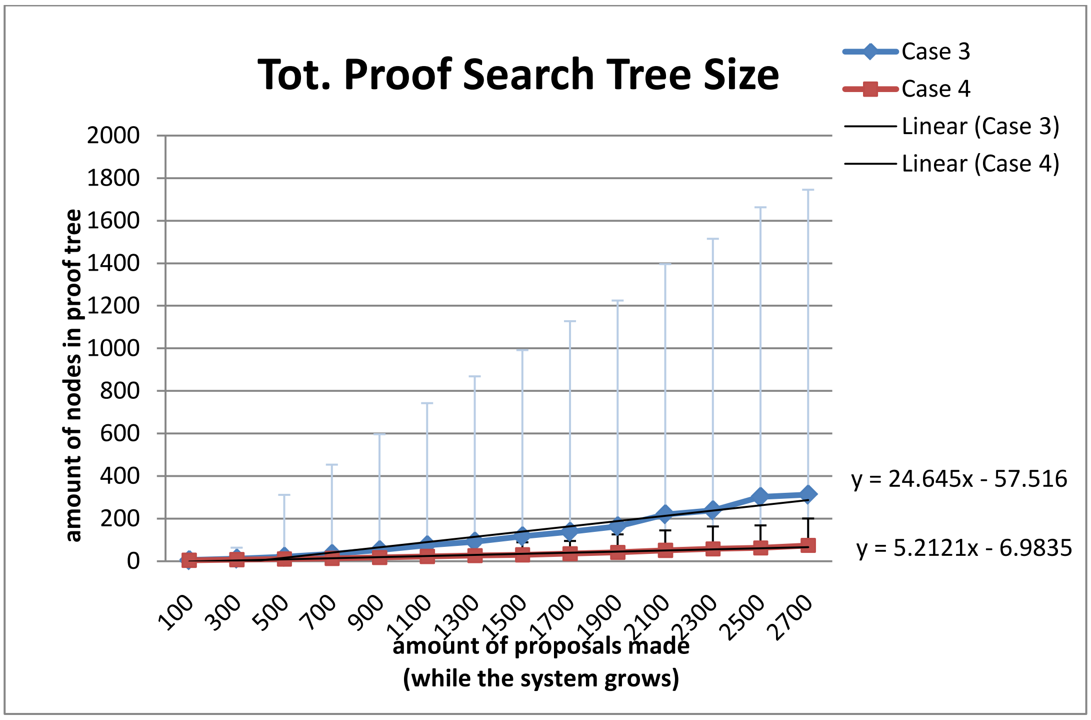


Figure 44. The y-axis here represents the total size of all proof search trees generated (i.e. one tree per sphere investigated) when a single proposal is processed. The x-axis is the same as before. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 3.

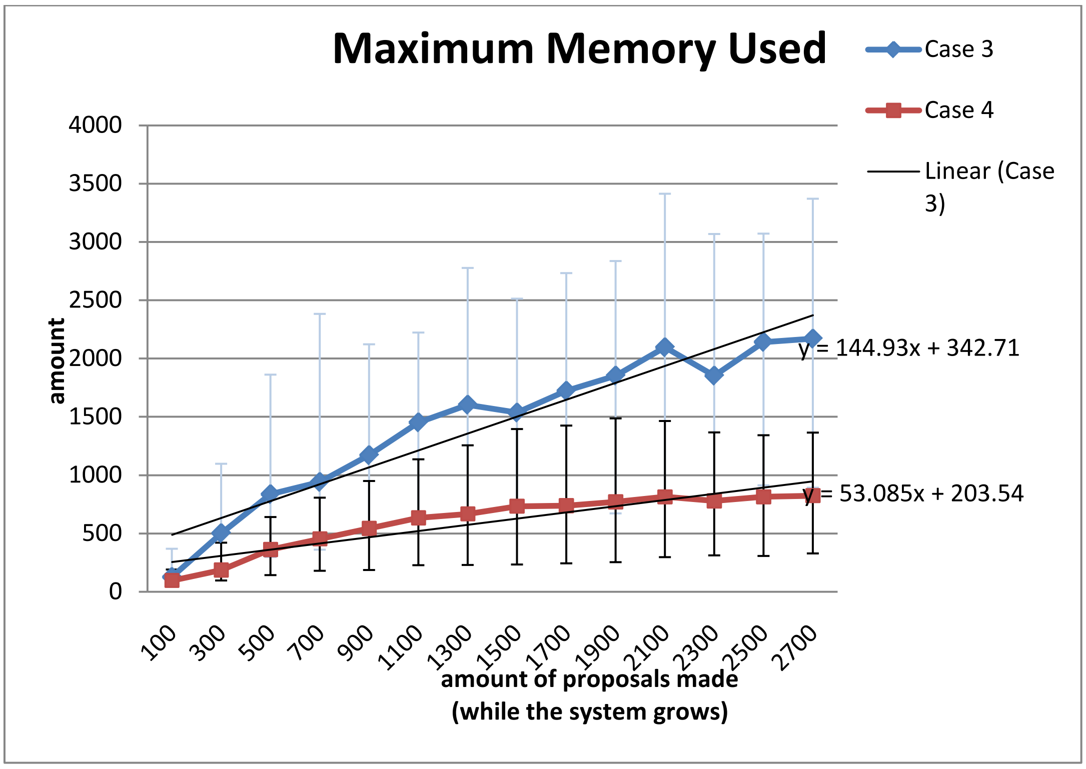


Figure 45. The y-axis shows the maximum memory used during the processing of a single proposal. If several spheres of consistency are investigated, then the one using most memory will influence the result. The error bars show the 25% and 75% percentiles for the two cases. The high error bars belong to case 1c.

The evolution of ontologies and mappings is based on proposals that are generated randomly with certain probabilities. However, these probabilities are chosen in such way that if the statistically expected size of ontologies 1, 3 and 5 are x, then the statistically expected size of ontologies 2, 4 and 6 are 10x. All the six ontologies have the branching factor 3. During the simulation, contexts can either propose to evolve their ontologies, or to propose mappings to the ontologies of contexts to which they have an active relation (this is exemplified in Scenario 1).

We learn from Figure 43 that the incremental effort to process a proposal grows linearly in both case 3 and 4, but that the growth of these linear functions is 7.8 times higher for case 3, i.e. when global consistency is maintained. If we compare with scenario 2.1 and its case 1 and case 2, we recall that it was 6.4, i.e. it has increased. However, the ratio that measures the effort considering total proof search size in Figure 44 is 4.75 (vs. 5.9 in scenario 2.1). Because the time measure is more important, we interpret this as an indication that in the three-organization topology the difference between maintaining full consistency and managed pair-wise consistency, is increased compared with the two-organization topology. The difference in memory use between these two topologies, is however small. Let us now discuss the consequences of these evaluations of various experimental scenarios.

7.5 General analysis of results

The general conclusions of the experimental evaluations are the following:

- If the inconsistency parameter p_c is moved sufficiently close to 2 (p_c=2 means that inconsistencies of all depths are fully allowed) then the incremental reasoning effort (e.g. the reasoning effort need to process a single proposal compared to the size of the knowledge in that sphere) converges towards constant time complexity. This means that even the size of the system grows this effort stays more or less the same. The cost one has to pay for this is that there is a certain probability of introducing inconsistencies in that sphere, but one knows that their size is above a certain threshold. This implies that queriesabout this knowledge set (in their simplest form, e.g. asking if a certain ontology relationships or mappings holds or not) will have a certain probability of being unsound.

- If we compare a scenario with several connected ontologies where full consistency is maintained vs. pair-wise consistency, then the variability of the time effort is much higher for the case where full consistency is maintained compared with the case when pair-wise consistency is maintained.

- When using the infrastructure mechanism with this particular simple logic (see chapter 3), the difference between maintaining full and pair-wise consistency does not lead to different computational time complexity, but the constant of the time complexity does change.

- It is sometimes useful to think in terms of interface ontologies, at the edge of an organization ontology network structure and used for communicating with other organizations. Core ontologies are then internal ontologies of an organization and are typically larger. A particularly interesting configuration of the infrastructure is to define pair-wise consistency in such a way that small interface ontologies are

shielded from big core ontologies when doing reasoning, by defining spheres of consistency that separates them, so that the core ontology of one organization has no *direct* contact with the ontologies of other organizations. However, by means of chains of ontology mappings and overlapping spheres of consistency, all ontologies are indirectly somehow connected.
The evaluation demonstrates that when the size difference increases between the large core ontologies and smaller interface ontologies, the system will benefit *even more* from using pair-wise consistency.

- It seems that the benefit of using proof-bounded consistency instead of full consistency is greater for a three-organization ontological topology compared to a two-organizational topology, so far as processing time effort is concerned. We note that the difference between these topologies is that the three organization topology has more ontologies and more complex structure than the two-organization topology.

- For this logic used the probability that either a forbidden contradiction (in the sense of "worse than allowed") or forbidden redundancy (defined in the corresponding way) should be detected or exist does converge towards zero when the inconsistency parameter p_c converges towards 2. This might also be true for other logics. But this was expected from the definitions (i.e. the endpoint of this convergence could be predicated but not the shape of the convergence curve).

Some general reflections:

- If a more expressive logic were used (e.g. a form of Description Logic that has exponential worst-time complexity), the computational benefit of using pair-wise consistency instead of global consistency would probably be bigger, because there is more benefit to bounding a higher time complexity and more expressive logics have worse time complexity. This is clear for the case when full consistency is used within all spheres of consistency (i.e. no proof-bounded consistency). However, it is more challenging to find an algorithm that would maintain incremental and proof-bounded consistency for description logics in general.

- The current prototype is actually very slow, in so far that it is using a lot of time to set up data structures etc. If the implementation is improved that time can be reduced.

To summarise, in this chapter we have investigated how the infrastructure mechanisms performs in several different scenarios that were created by simulations. We have learnt that it makes it feasible to automatically maintain consistency between several interconnected ontologies, and that it does make the process more scalable.

Chapter 8 Proofs of infrastructure mechanism properties

In this chapter we prove four theorems about the properties of the infrastructure mechanism. The first three ones are related to the logical reasoning whereas the last one is concerned with the process model.

We first look at Table 3 below and it exhaustively investigates all cases of composing two relations in the form of $R \circ R' = R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$ where R and R' are either ontology mappings or ontology relations.

Table 3. The result of calculating $R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$. It implicitly assumed that $R(A_i, B_j)$ is created in state s and $R'(B_j, C_k)$ is created in state s', whereas $R''(A_i, C_k)$ is created in state s''=max(s, s').

R= / R'=	COR(A_i, B_j)	IS (A_i, B_j)	IS2 (A_i, B_j)	COMPATIBLE (A_i, B_j)	DISJOINT (A_i, B_j)
COR (B_j, C_k)	COR (A_i, C_k)	IS (A_i, C_k)	IS2 (A_i, C_k)	COMPATIBLE (A_i, C_k)	DISJOINT (A_i, C_k)
IS (B_j, C_k)	IS (A_i, C_k)	IS (A_i, C_k)	ε	COMPATIBLE (A_i, C_k)	ε
IS2 (B_j, C_k)	IS2 (A_i, C_k)	ε	IS2 (A_i, C_k)	ε	DISJOINT (A_i, C_k)
COMPATIBLE (B_j, C_k)	COMPATIBLE (A_i, C_k)	ε	COMPATIBLE (A_i, C_k)	ε	ε
DISJOINT (B_j, C_k)	DISJOINT (A_i, C_k)	DISJOINT (A_i, C_k)	ε	ε	ε

We have calculated the entries in Table 3 by using the mechanical rewrite rules in Chapter 4. E.g. in section 4.10 we show that COR ∘ COR = COR and this corresponds to one of the entries in the table above.

We now look at the theorems.

Theorem 1. This theorem states that:

Suppose

R= Cor (A_i, B_j)| Is (A_i, B_j)| Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j)

$R' = Cor(B_j, C_k) \mid Is(B_j, C_k) \mid Is2(B_j, C_k) \mid \textsc{Disjoint}(B_j, C_k) \mid \textsc{Compatible}(B_j, C_k)$
$R'' = Cor(A_i, C_k) \mid Is(A_i, C_k) \mid Is2(A_i, C_k) \mid \textsc{Disjoint}(A_i, C_k) \mid \textsc{Compatible}(A_i, C_k)$

and
$S = Cor(A_i, C_k) \mid Is(A_i, C_k) \mid Is2(A_i, C_k) \mid \textsc{Disjoint}(A_i, C_k) \mid \textsc{Compatible}(A_i, C_k)$

Then the following two are equivalent:

- $R(A_i, B_j) \wedge R'(B_j, C_k) \vDash R''(A_i, C_k)$ and there is no $S(A_i, C_k)$ such that $S(A_i, C_k) \vDash R''(A_i, C_k))$
- $(R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k))$

I.e. every relation (as defined in §3.4) that semantically follows from the semantic conjunction of two relations (and there is no more general relation that is true), this relation is derivable as shown in Table 3, and every relation that is derivable[4] as shown in Table 3 (being a conjunction of two other relations) is also semantically true (i.e. the most general relation that is true).

Theorem 2. The algorithms used for reasoning provide decision procedures for the following proof tasks:

- Proof task 1. Calculating if a proposed change introduces a contradiction.
- Proof task 2. Calculating if a proposed change introduces a redundancy.
- Proof task 3. Calculating if a proposed change introduces a change that neither generates a contradiction or redundancy.

Theorem 3. The process execution of the described in Chapter 5 will always terminate, when it is activated by the any of the two proposals below assuming that contexts are responsive during the process. When it terminates it will either remove knowledge contradicted by these proposals, or accept and perform them, or reject them:

- c_i : PROPOSE (add_ontorel(m, c_j, d_j))
- c_i : PROPOSE (add_mapping(m, c_i, c_k))

[4] The rewrite rules and the reasoning algorithms in sections §4.7 to §4.12 are used in the mechanical derivation.

in finite time, in accordance to the criteria in theorem 2 and maintain the defined consistency constraints. It will only reject the proposals if there is a violation of the consistency constraints.

8.1 Proof of Theorem 1

In §4.10 we have defined the rules for calculating the outcome of this combination:

$R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$

where R, R' and R'' are either relations between concepts in an ontology or mappings between concepts in two different ontologies, and where A is a concept in ontology *i*, B is a concept in ontology *j*, and C is a concept in ontology *k*. Let us assume that R is created in state *s*, and R´ is created in state s´ whereas R'' is created in state s´´. If both relationships are within the same ontology then $i=j=k$. If R is a relation within an ontology and R' a mapping between two ontologies then R'' is a mapping between two ontologies and $i=j\neq k$. So we will now focus on the mechanical computation of the equation above (according to the algorithms in chapter 4) and verify that it adheres to the semantics defined in chapter 3. The first thing we will do is to list the exhaustive list of all possibilities that R, R' and R'' can have given our focus on a limited language as defined in §3.4. All these combinations are shown in Table 3.

A comment about notation: In this chapter we use variable z´ instead of z to denote the state of an expression, because variable z is sometimes used as an instance variable.

8.1.1 Combining States when Calculating $R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$

In the analysis of all the cases in Table 3 we will not focus so much on which state the relations are created. The reason for this is that this follows a regular pattern that is very similar for all the cases.

Given that $R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$
If $R(A_i, B_j)$ is created in state *s* and $R'(B_j, C_k)$ is created *s'* then $R''(A_i, C_k)$ is created in state max(*s*, *s'*). This is easy to see for these cases:
R= Cor (A_i, B_j)| Is (A_i, B_j)| Is2 (A_i, B_j) | DISJOINT (A_i, B_j)

R'= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k)

The reason is that all of these relations use a syntax of the form

$$N_s(\exp(A_i(x_i), B_j(y_j)) \wedge G_s(\exp(A_i(x_i), B_j(y_j))$$

Let's assume *s'>s.*

When two such relations are combined we get a combination like this:

$$\begin{aligned}
&(N_s(\exp(A_i(x_i), B_j(y_j))) \wedge G_s(\exp(A_i(x_i), B_j(y_j)))) \wedge\\
&(N_{s'}(\exp'(B_j(y_j), C_k(z_k))) \wedge G_{s'}(\exp'(B_j(y_j), C_k(z_k))))\\
&\Rightarrow\\
&N_s(\exp(A_i(x_i), B_j(y_j))) \wedge N_{s'}(\exp'(B_j(y_j), C_k(z_k))) \wedge\\
&N_s(\exp(A_i(x_i), B_j(y_j))) \wedge G_{s'}(\exp'(B_j(y_j), C_k(z_k))) \wedge\\
&G_s(\exp(A_i(x_i), B_j(y_j))) \wedge N_{s'}(\exp'(B_j(y_j), C_k(z_k))) \wedge\\
&G_s(\exp(A_i(x_i), B_j(y_j))) \wedge G_{s'}(\exp'(B_j(y_j), C_k(z_k)))\\
&\Rightarrow\\
&G_s(\exp(A_i(x_i), B_j(y_j))) \wedge N_{s'}(\exp'(B_j(y_j), C_k(z_k))) \wedge\\
&G_s(\exp(A_i(x_i), B_j(y_j))) \wedge G_{s'}(\exp'(B_j(y_j), C_k(z_k)))\\
&\Rightarrow\\
&\forall(st \in S)(L(s,st) \to V_{i,st}[\exp(A_i(x_i), B_j(y_j)) |_{z'=st}] = 1) \wedge\\
&V_{i,s'}[\exp'(B_j(y_j), C_k(z_k)) |_{z'=st'}]) = 1 \wedge\\
&\forall(st \in S)(L(s,st) \to V_{i,st}[\exp(A_i(x_i), B_j(y_j)) |_{z'=st}] = 1) \wedge\\
&\forall(st' \in S)(L(s',st') \to V_{i,st'}[\exp'(B_j(y_j), C_k(z_k)) |_{z'=st'}] = 1)\\
&\Rightarrow\\
&V_{i,s'}[\exp(A_i(x_i), B_j(y_j)) \wedge \exp'(B_j(y_j), C_k(z_k)) |_{z'=st'}] = 1) \wedge\\
&\forall(st' \in S)(L(s',st') \to V_{i,st'}[\exp(A_i(x_i), B_j(y_j)) \wedge \exp'(B_j(y_j), C_k(z_k)) |_{z'=st'}] = 1)\\
&\Rightarrow\\
&N_{s'}(\exp(A_i(x_i), B_j(y_j) \wedge \exp'(B_j(y_j), C_k(z_k))) \wedge G_{s'}(\exp(A_i(x_i), B_j(y_j)) \wedge \exp'(B_j(y_j), C_k(z_k)))\\
&\Rightarrow\\
&N_{s'}(\exp''(A_i(x_i), C_k(z_k))) \wedge G_{s'}(\exp''(A_i(x_i), C_k(z_k)))
\end{aligned}$$

This calculation has several times used the fact that *s´>s*, otherwise this conclusion would not have been possible to reach. This is equivalent to saying that the resulting relation R''(A_i, C_k) will belong to state max(*s,s´*).

Let us now review again the equation

R(A$_i$, B$_j$) ∧ R'(B$_j$, C$_k$) ⇒ R"(A$_i$, C$_k$)

and assume again that R(A$_i$, B$_j$) is created in state *s* and R'(B$_j$, C$_k$) is created *s'*.

However, now

R= Cor(A$_i$, B$_j$) | Is(A$_i$, B$_j$) | Is2(A$_i$, B$_j$) | DISJOINT(A$_i$, B$_j$)

R'= COMPATIBLE(B$_j$, C$_k$)

$$
\begin{aligned}
&N_s(\exp(A_i(x_i), B_j(y_j))) \wedge G_s(\exp(A_i(x_i), B_j(y_j))) \wedge \\
&F_{s'}(\exp'(B_j(y_j), C_k(x_k))) \\
&\Rightarrow \\
&V_{(i,j),s}[\exp(A_i(x_i), B_j(y_j))|_{z'=s}]) = 1 \wedge \\
&\forall(st \in S)(L(s,st) \rightarrow V_{(i,j),st}[\exp(A_i(x_i), B_j(y_j))|_{z'=st}] = 1) \wedge \\
&\exists(st' \in S)(L(s',st') \rightarrow V_{(i,j),st'}[\exp'(B_j(y_j), C_k(x_k))|_{z'=st'}] = 1) \\
&\Rightarrow \\
&\forall(st \in S)(L(s-1,st) \rightarrow V_{(i,j),st}[\exp(A_i(x_i), B_j(y_j))|_{z'=st}] = 1) \wedge \\
&\exists(st' \in S)(L(s',st') \rightarrow V_{(i,j),st'}[\exp'(B_j(y_j), C_k(x_k))|_{z'=st'}] = 1) \\
&\Rightarrow (\text{assume } s' \geq s-1) \\
&\exists(st' \in S)(L(s',st') \rightarrow V_{(i,j),st'}[\exp(A_i(x_i), B_j(y_j)) \wedge \exp'(B_j(y_j), C_k(x_k))|_{z'=st'}] = 1) \\
&\Rightarrow \\
&\exists(st' \in S)(L(s',st') \rightarrow V_{(i,j),st'}[\exp''(A_i(x_i), C_k(x_k))|_{z'=st'}] = 1)
\end{aligned}
$$

So in this situation we are given some kind of answer (later sections will show what answer). If however, $s' < s-1$ then we construct a counter-model where exp' holds in a state before all the states where s holds, i.e. there will be no state where exp and exp' hold at the same time (what we call exp´´). This means combining the relations when $s' < s-1$ will lead to R(A$_i$, B$_j$) ∧ R'(B$_j$, C$_k$) ⇒ ε

If we swap the order of R and R' (i.e. R= COMPATIBLE(A$_i$, B$_j$)) and if it is $s' < s-1$ then again R(A$_i$, B$_j$) ∧ R'(B$_j$, C$_k$) ⇒ ε

So we will keep the following in mind:

When R(A$_i$, B$_j$) ∧ R'(B$_j$, C$_k$) ⇒ R"(A$_i$, C$_k$)

and R= COMPATIBLE and is created in state s' whereas R' is created in state s, or

R´= COMPATIBLE and is created in state s' whereas R is created in state s,

then if $s' < s-1$ then R"= ε.

8.1.2 The distinction between ontology relations and mappings when calculating $R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$

This is again something we want to investigate now for all cases in Table 3 instead of repeating this over and over again for all cases that occur.

The relations in Table 3 refer to ontology mappings if $i \neq j$, or to ontology relations if $i=j$.

If we combine two ontology mappings $R(A_i, B_j) \wedge R'(B_j, C_k) \Rightarrow R''(A_i, C_k)$ we will get a new ontology mapping if $i \neq k$. If we combine two ontology relations $R(A_i, B_i) \wedge R'(B_i, C_i) \Rightarrow R''(A_i, C_i)$ we get a new ontology relation. If we combine an ontology mappings and an ontology relation we get a new ontology mapping: $R(A_i, B_j) \wedge R'(B_j, C_j) \Rightarrow R''(A_i, C_j)$. Therefore, by following these rules we always now if the end-result is a mapping or ontology relationship.

8.1.2.1 Solution: Choose rich semantics (ontology mappings) and project to simpler when needed

The question remains if we have to repeat all proof for all these cases, considering that a relation $R(A_i, B_j)$ has a somewhat different extended syntax and semantics, if $i=j$ and it is an ontology relationship compared to if $i \neq j$ and it is an ontology mapping. The answer is that it is enough to investigate the case when both $R(A_i, B_j)$ and $R'(B_j, C_k)$ are ontology mappings, because they then used the "richest" semantics, in the following way.

In those cases with simply transform all ontology mappings into corresponding ontology relations and the instance variables x_i, y_j and z_k are mapped into x_i, y_i and z_i in ontology i. What is a counter-model in the original theory (using several ontologies) will still be a counter-model when it is translated together with the theory into a single ontology in this way.

8.1.2.2 A proof pattern used

This proof pattern will be used a few times, so we show it now.

Assume that $\Theta(A(x), B(y)) \wedge \Theta'(B(y), C(z)) \to \Theta''(A(x), C(z))$

It then holds that

$\forall xy(R(x,y) \to \Theta(A(x),B(y))) \wedge \forall yz(R(y,z) \to \Theta'(B(y),C(z)))$
$\Rightarrow$
$\forall xz(R(x,z) \to (\Theta''(A(x),C(z)))$

The proof is as follows.

$\forall xy(R(x,y) \to \Theta(A(x),B(y))) \wedge \forall yz(R(y,z) \to \Theta'(B(y),C(z)))$
$\Rightarrow$
$\forall xyz(R(x,y) \to \Theta(A(x),B(y)) \wedge R(y,z) \to \Theta'(B(y),C(z)))$
$\Rightarrow$
$\forall xyz((\neg R(x,y) \vee \Theta(A(x),B(y))) \wedge (\neg R(y,z) \vee \Theta'(B(y),C(z))))$
$\Rightarrow$
$\forall xyz((\neg R(x,y) \wedge \neg R(y,z)) \vee (\neg R(x,y) \wedge \Theta'(B(y),C(z))) \vee$
$(\Theta(A(x),B(y)) \wedge \neg R(y,z)) \vee (\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z))))$
$\Rightarrow$
$\forall xyz((\neg R(x,y) \wedge \neg R(y,z)) \vee \neg R(x,y) \vee \neg R(y,z)$
$\vee(\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z))))$
$\Rightarrow$
$\forall xyz(\neg R(x,y) \vee \neg R(y,z) \vee (\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z))))$
$\Rightarrow$
$\forall xyz(\neg(R(x,y) \wedge R(y,z)) \vee (\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z))))$
$\Rightarrow$ (transitivity of R(x,y))
$\forall xyz(\neg(R(x,z)) \vee (\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z))))$
$\Rightarrow$
$\forall xyz(R(x,z) \to (\Theta(A(x),B(y)) \wedge \Theta'(B(y),C(z)))$
$\Rightarrow$
$\forall xz(R(x,z) \to (\Theta''(A(x),C(z)))$

Notice that $\Theta(A(x),B(y))$ and $\Theta'(B(y),C(z))$ are Boolean functions and $\Theta''(A(x),C(z))$ is their conjunction without the predicate $B(y)$.

All the proof details of theorem 1 are carried out in Appendix A, so the reader is advised to read Appendix A at this stage.

8.2 Proof of Theorem 2

We will first prove the soundness and completeness of the procedure for the first proof task. Before we can do this we will briefly define contradiction between relations and a different use of the conjunction.

8.2.1 Defining contradiction

In §4.11 we defined how negation is applied in a mechanical way as a part of the derivation procedure, and we see that ¬DISJOINT (A_i, C_k) = COMPATIBLE (A_i, C_k) and vice versa.

Contradiction can only occur in the follow way:

$$\frac{\text{DISJOINT}\ (A_i, C_k) \qquad \text{COMPATIBLE}\ (A_i, C_k)}{\text{contradiction.}}$$

I.e. this is an application of the third rules in Figure 46. This rule is symmetric, so COMPATIBLE (A_i, C_k) could equally well occur on the left side.

However, as part of future research (see §9.2.7) it would be advisable to add support for discovering contradictions that occur when instances are added after ontology relations and mappings have been defined. Then, it would be possible for the system find contradictions even for the relations COR(A_i, C_k), Is (A_i, C_k) and Is2 (A_i, C_k).

$$\frac{R(A_i, B_j) \qquad R'(B_j, C_k)}{R''(A_i, C_k)}$$

$$\frac{R(A_i, B_j)}{R'(B_j, A_i)}$$

$$\frac{R(A_i, B_j) \qquad \neg R(A_i, B_j)}{\text{Contradiction}}$$

Figure 46. Three rules used: conjunction, variable permutation and reaching contradiction.

8.2.2 Proof Task 1. Calculating if a proposed change introduces a contradiction.

We are restricting the semantics to only use the relations R defined in §3.4 and the proof patterns described in §4.6.
The "proof structure" below investigate the structure of all possible proofs that could be created, but we will first look at the rules that actually are used by the reasoning algorithms. Figure 46 shows the three operations used by the reasoning algorithms: conjunction ($\wedge$),

permutation of variables and detection of contradiction (that also can include using negation ($\neg$)). §8.2.1 has defined how contradictions can occur. We will not discuss variable permutation any further, because it is a technical detail – it is enough to say that it is included in the other operations, so when the reasoning system is looking for relations matching the reasoning rules it also does use variable permutation if needed. We will now investigate which proofs that are possible in general.

Proof structure

Let us assume that there is a set of existing relations $R_1, R_2, ..., R_n$ and that an new relation G has been added that creates a contradiction. G creates a contradiction iff there is a finite sequence where these four rules on the existing relations

Rule 1.

$$\frac{\Gamma \vdash R(A_i, B_j) \quad \Gamma \vdash R'(B_j, C_k)}{\Gamma \vdash R''(A_i, C_k)}$$

(where R" is calculated using Table 3)

Rule 2.

$$\frac{\Gamma \vdash R(A_i, B_j)}{\Gamma \vdash R'(B_j, A_i)}$$

(where R' is the antisymmetric relation to R)

Rule 3.

$$\frac{\text{If } \Gamma \vdash R(A_i, B_j) \quad \Gamma \vdash \neg G(A_i, B_j)}{\Gamma \vdash \text{contradiction}}$$

(if the rule matches the case in §8.2.1)

Rule 4.

$$\frac{\Gamma \vdash IS(A_i, B_j) \quad \Gamma \vdash IS2(A_i, B_j)}{\Gamma \vdash COR(A_i, B_j)}$$

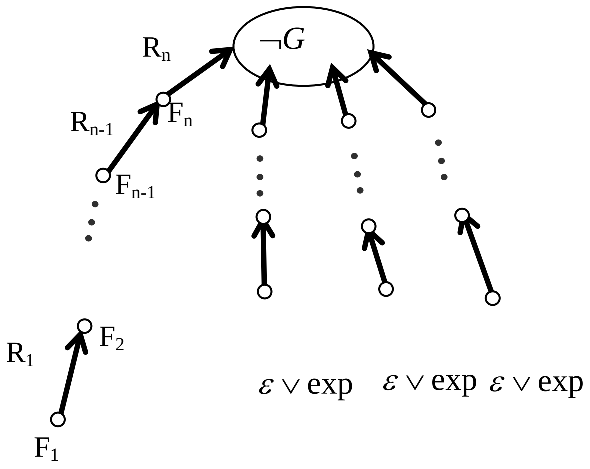


Figure 47. The newly proposed mappings is called G and $R_n, R_{n-1}, ..., R_1$ are the mappings that are used as rules in this proof search tree, where F_1 is true.

Notice that Algorithm A uses these rules in a backward fashion, because it starts with rule 3 and tries to find a matching relation that would generate the contradiction it is looking for.

8.2.2.1 Soundness and completeness

Let us now investigate soundness and completeness for this particular case when the negation of a newly proposed mapping G can inferred from a minimal set of existing mappings R_n, R_{n-1},..., R_1. The algorithm is described in §4.12.2 and called Algorithm A. We investigate the case when the level of consistency within a sphere of consistency is not bound, p_c=1. This is illustrated in Figure 47. The existing relations are used as rules. If we go back to the definition of the rule application procedure we see that

$F_1 \wedge R_1 \to F_2$

$F_1 \wedge R_1 \wedge R_2 \to F_3$ (because $\mathrm{F}_2 \wedge R_2 \to F_3$)

.

.

$F_1 \wedge \bigwedge_{i=1..n} R_i \to \neg G$

From the point of the view of the algorithm, it is doing breadth-first search and creating a proof search tree. It is using rule 1 and (implicitly) rule 2 from Figure 46 and rule 3 to end a proof. If all paths except one end with an ε or any expression that still will be processed (this is called exp in Figure 47), then that single path stops at an expression F_1 that we can prove is true (because it is a tautology or follows from one of the existing mappings). Then, from a logical point of view, we have a set of mappings whose conjunction implies the negation of a proposed mapping:

$$\bigwedge_{i=1..n} R_i \to \neg G$$

There is, however, a discrepancy between the three rules that Algorithm A is actually using (see Figure 46 and the four rules listed above. The essence of that discrepancy is rule 4 that says that IS(A_i, B_j) ∧ IS2(A_i, B_j) ⇒ COR(A_i, B_j). In reality, this rules is never needed for the following reason: The only contradiction that can occur are COMPATIBLE(A_i, B_j) and DISJOINT(A_i, B_j) or a big set of relations that can be reduced to these two using composition.

Let us now use a graph-like metaphor and recall the contents of Table 3. If such a proof search tree has to be propagated over these two relations: $IS(A_i, B_j) \wedge IS2(A_i, B_j)$ in order to make a proof, then it is enough if it propagates over one of these relations. E.g. COR o DISJOINT ⇒ DISJOINT, but also IS o DISJOINT ⇒ DISJOINT, and COR o COMPATIBLE ⇒ COMPATIBLE, but also IS2 o COMPATIBLE ⇒ COMPATIBLE. In the case that DISJOINT or COMPATIBLE are composed with COR, we again can find either IS or IS2 that gives the same result. Therefore, because we use this particular logic with these particular reasoning rules, completeness does not need rule 4 from §8.2.2 above.

The procedure is therefore sound, become we know that every derivation step (e.g. using conjunction) is sound (see Theorem 1). It is also complete because if there is a proof it will find it because it uses breath first search. Section 8.3 explains why the procedure will terminate. Also, if there are several different proofs (i.e. sets of existing mappings) then it will find the shortest proof (i.e. smallest set or relations).

8.2.3 The case of bounded consistency

In algorithm A, there is a line saying " Let $d=(2-p_c)(tot-1)+2$". This uses the consistency parameter p_c and an estimate of the maximum size of the amount of elements (ontology relations or mappings) within that sphere of consistency. Based on these inputs it calculates the absolute minimum depth that a contradiction can have, because the short contradictions that are more obvious are seen as more dangerous than those that are more subtle because they require a longer proof.

Later Algorithm A says "If l>=d then return false" where l is the depth of the proof search tree. Then the algorithm has verified there are no proofs of size d-1, i.e. because we had decided that that we are not interested in proofs of size d or larger.

This is the whole procedure. The user selects a consistency parameter p_c and estimate of the total maximum size *tot*. From this, the absolute length *d* is calculated and guarantees that once size *tot* has been reach, that consistency parameter p_c will hold then. On the way to that to the estimated maximum size p_c will also hold, but $p_{actual}<p_c$ then because the *d* is fixed (future versions of such system could find ways of modifying *d* during run-time).
Therefore, if a proposal is given to algorithm A, if it finds a proof of contradiction shallower than *d,* it will notify the user and the proposed change will not be done. If it, however, does not find a proof of contradiction shallower than depth *d,* it will not continue searching and

there might potentially be a contradiction of depth *d* or larger that remains in the sphere of consistency. However, next time it does process a proposal to add a relation, it will search for contradictions within depth *d* around (i.e. in the network vicinity) that proposal, so if that change would contribute to making a contradiction of size less than *d*, it will find it.
We have therefore shown how Algorithm A maintains the desired level of consistency p_c.

8.2.4 Proof task 2. Calculating if a proposed change introduces a redundancy.

The algorithmic task of solving this process is similar to proof task 1. So the description will be somewhat briefer. Figure 48 shows the operations used by the algorithm (this time called Algorithm B), conjunction ($\wedge$), permutation of variables and detection of redundancy (that also can include using negation ($\neg$) or be inferred from a self-contradiction). The following section defines when the redundancy rule is triggered.

$$\frac{R(A_i, B_j) \qquad R'(B_j, C_k)}{R''(A_i, C_k)}$$

$$\frac{R(A_i, B_j)}{R'(B_j, A_i)}$$

$$\frac{R(A_i, B_j) \qquad \neg R(A_i, B_j)}{\text{Redundancy}}$$

$$\frac{S(A_i, A_i) \qquad \text{(S=self-contrad.)}}{\text{Redundancy}}$$

Figure 48. Four rules possible: conjunction, variable permutation and reaching redundancy (two ways).

8.2.4.1 Defining Redundancy and Self-contradiction rules

Firstly, Rule 3 below fires iff rule 3 in § 8.2.2 would have fired, but in this case this proves that G is redundant. It is explained below why this is the case, but the main reason is that the forward reasoning includes ¬G in the very start.

Secondly, Rule 3b will fire iff $S(A_i, A_i)$ is one of the following self-contradictions:
¬IS(A_i, A_i), ¬IS2(A_i, A_i), ¬COR(A_i, A_i), ¬COMPATIBLE(A_i, A_i), DISJOINT(A_i, A_i)

8.2.4.2 The Proof Structure

Above we have defined which relations that contradict each other, and self-contradiction. We will see that the main difference is that we apply the rules in forward direction here, compared to the backward direction used in proof task 1.

Proof structure

Let us assume that there is a set of existing relations $R_1, R_2, ..., R_n$ and a new relation G. It creates a redundancy iff there is a proof of redundancy by applying a sequence of these rules to these assumptions.

Rule 1.

$$\frac{\Gamma\vdash R(A_i, B_j) \qquad \Gamma\vdash R'(B_j, C_k)}{\Gamma\vdash R''(A_i, C_k)}$$

(where R" is calculated using Table 3. Notice that R could also be $\neg G(A_i, B_j)$)

Rule 2.

$$\frac{\Gamma \vdash R(A_i, B_j)}{\Gamma \vdash R'(B_j, A_i)}$$

(where R' is the symmetric relation to R)

Rule 3.

If $\dfrac{\Gamma\vdash R(A_i, B_j) \qquad \Gamma\vdash\neg R(A_i, B_j)}{\Gamma\vdash \text{redundancy}}$

Rule 3b

If $\dfrac{\Gamma\vdash S(A_i, A_i)}{\Gamma\vdash \text{redundancy}}$

(Where $S(A_i, A_i)$ is a self-contradiction as defined in §8.2.4.1)

Rule 4.

$\Gamma \vdash IS(A_i, B_j) \qquad \Gamma \vdash IS2(A_i, B_j)$

$\Gamma \vdash COR(A_i, B_j)$

Notice that Algorithm B uses these rules in a forward fashion *always* starting with Rule 1 containing $\neg G(A_i, B_j)$ as one of the two relations. It is because of this starting point that an encountered contradiction actually proves that G is redundant.

8.2.5 Listing all rule instances

We will first list all instances of rule 3 in section 8.2.4, and they are:

If $\Gamma \vdash R(A_i, B_j) \qquad \Gamma \vdash \neg R(A_i, B_j)$

$\Gamma \vdash$redundancy

(where R = $\text{COR}(A_i, B_j)$ | $\text{IS}(A_i, B_j)$ | $\text{IS2}(A_i, B_j)$ | $\text{COMPATIBLE}(A_i, B_j)$ | $\text{DISJOINT}(A_i, B_j)$)

If $\Gamma \vdash \text{COR}(A_i, B_j) \qquad \Gamma \vdash \neg\, \text{IS}(A_i, B_j)$

$\Gamma \vdash$redundancy

If $\Gamma \vdash \text{COR}(A_i, B_j) \qquad \Gamma \vdash \neg\, \text{IS2}\ (A_i, B_j)$

$\Gamma \vdash$redundancy

If $\Gamma \vdash \text{COMPATIBLE}\ (A_i, \neg B_j) \qquad \Gamma \vdash \text{IS}(A_i, B_j)$

$\Gamma \vdash$redundancy

If $\Gamma \vdash \text{COMPATIBLE}\ (\neg A_i, B_j) \qquad \Gamma \vdash \text{IS}(A_i, B_j)$

$\Gamma \vdash$redundancy

If $\Gamma \vdash$ COMPATIBLE (A_i, B_j) $\qquad$ $\Gamma \vdash$ DISJOINT(A_i, B_j)

$\Gamma \vdash$redundancy

The reader is reminded that the conclusion "redundancy" refers to the original proposal G (i.e. "G is redundant"), because all this rules just show the last step in a reasoning chain and are actually contradictions as such.

Table 3 indeed shows the main cases for rule 1, but during reasoning about redundancy this rules instances also have to be taken into account:

$\Gamma \vdash \neg R(A_i, B_j)$ $\qquad$ $\Gamma \vdash$ COR(B_j, C_k)

$\Gamma \vdash \neg R(A_i, C_k)$

(where R = COR(A_i, B_j) | IS(A_i, B_j) | IS2(A_i, B_j) | COMPATIBLE(A_i, B_j) | DISJOINT(A_i, B_j)

$\Gamma \vdash \neg$IS(A_i, B_j) $\qquad$ $\Gamma \vdash$ IS2(B_j, C_k)

$\Gamma \vdash$ COMP$(A_i, \neg C_k)$

$\Gamma \vdash \neg$IS(A_i, B_j) $\qquad$ $\Gamma \vdash$ DISJOINT(B_j, C_k)

$\Gamma \vdash$ COMP(A_i, C_k)

$\Gamma \vdash \neg$IS2(A_i, B_j) $\qquad$ $\Gamma \vdash$ IS(B_j, C_k)

$\Gamma \vdash$ COMP$(\neg A_i, C_k)$

$\Gamma \vdash \neg$IS2(A_i, B_j) $\qquad$ $\Gamma \vdash$ DISJOINT(B_j, C_k)

$\Gamma \vdash$ COMP($\neg A_i$, $\neg C_k$)

We show soundness and completeness for the rules in Table 3 (See Chapter 8 and Appendix A), and using a similar method one can easily prove soundness for these few rules above as well.

8.2.6 Which proofs are possible?

The proofs possible are similar to the ones in Figure 46 but rules are now used in a forward direction instead (see Figure 49). So e.g. now proofs like this are possible:

$\neg G \wedge R_1 \wedge R_2 \rightarrow \neg R_3$

8.2.7 Soundness and completeness

Let us now investigate soundness and completeness for this particular case when the newly proposed mapping G can be inferred from a minimal set of existing relations R_n, R_{n-1},…,R_1.

We refer to Algorithm B described in §4.12.4.

Let us now investigate soundness and completeness for this particular case when a newly proposed mapping G can be inferred from a minimal set of existing mappings R_n, R_{n-1},…,R_1 or is contradicting a set of existing mappings (there is a third case that will be investigated in the next section). This is illustrated in Figure 49 . The existing mappings are used as rules. If we go back to the definition of the rule application procedure we see that:

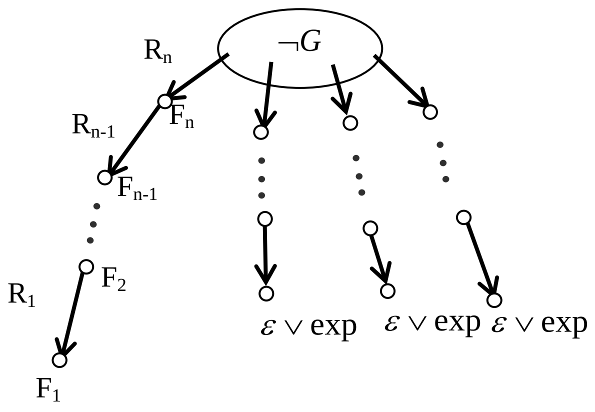


Figure 49. The newly proposed mapping is called G and is negated, and R_n, R_{n-1},…,R_1 are the mappings that are used as rules in this proof search tree, where F_1 has to be false.

$$\neg G \wedge R_n \rightarrow F_n$$
$$\neg G \wedge R_n \wedge R_{n-1} \rightarrow F_{n-1} \text{ (because } F_n \wedge R_{n-1} \rightarrow F_{n-1}\text{)}$$
.

.

$$\neg G \wedge \bigwedge_{i=1..n} R_i \rightarrow F_1$$

From the point of the view of the algorithm, it is doing breadth-first search and creating a proof search tree. Termination is clear from the analysis in §8.3. If all paths except one end with an ε expression (or are in the process of being investigated - denoted as "exp" in Figure 49), then these paths terminate (or would still have been investigated) and that single path stops at an expression F_1 that we can prove is false (because it is a negated tautology or contradicts one of the existing mappings). Then, from a logical point of view, we have the following (when F_1 is false):

$$\neg G \wedge \rightarrow \bigwedge_{i=1..n} R_i \Rightarrow false$$
$$\neg(\neg G \wedge \Rightarrow \bigwedge_{i=1..n} R_i)$$
$$G \vee \neg \bigwedge_{i=1..n} R_i$$

There the procedure is therefore sound, and in Theorem 1 we showed that every derivation step is sound.

8.2.8 Incompleteness

The procedure would have been complete if all proofs had structure outlined above, because if there is a proof it will find it. Also, if there are several different proofs (i.e. sets of existing mappings) then it will find the shortest proof (i.e. smallest set). However, if one uses a special construction of the following form the proof procedure won't find them:

$$\frac{\Gamma \vdash IS(A_i, B_j) \quad \text{and} \quad \Gamma \vdash IS2(A_i, B_j)}{\Gamma \vdash COR(A_i, B_j)}$$

Because this is semantically true, while the mechanical procedure does not use this rule. If this construct is avoided there is still completeness, but if it is used, then the system won't be able to detect some redundancies.

8.2.9 The case of bounded redundancy

If we repeat the argument of §8.2.3 but consider redundancy instead of consistency, and p_r instead of p_c then we reach the corresponding conclusion for bounded redundancy. However, it only holds if we refrain from using the IS/IS2 construct in §8.2.8.

8.2.10 Proof task 3. Calculating if a proposed change introduces a change that neither generates a contradiction or redundancy.

This is actually only a combination of the two earlier algorithms: algorithm A (from proof task 1) and algorithm B (from proof task 2). If algorithm A answers yes or algorithm B answers yes, then the answer for this proof task will be no. But if both algorithm A and B answer no (i.e. a proposed change neither creates a contradiction nor redundancy), then this proof task is given the answer yes – the relation is new.

8.3 Termination

We now clarify why the algorithms for the three proof tasks in chapter 4.6.1. (see above) will terminate for any valid input.
For the case of "Proof Task 1. Calculating if a proposed change introduces a contradiction." The algorithm does breadth-first search and has looping prevention. Because the maximum depth of this proof search tree is the amount of relations in the sphere of consistency where the reasoning is done, the reasoning is bounded depth-wise.
How can we be sure that the branching factor is limited? We know this because the maximum amount of remaining knowledge elements (ontology relations or mappings within a sphere of consistency) is the upper found on how many branches there can be.
Because the ontologies are connected with mappings, the relations can form loops. How can we then be sure that the reasoning terminates? We can be sure of this, because there is a

loop-prevention mechanism in the reasoning algorithm that will not add knowledge to the proof search tree that already exists there.
For the case of finding redundancy the same arguments apply, because the branching factor and depth are both limited.

8.4 Theorem 3

At this stage we already know that the reasoning terminates, but we just want to be sure that the processing of proposals terminates.

Theorem 3 states that:

Theorem 3. The process execution of the described in Chapter 5 will always terminate, when it is activated by the any of the two proposals below assuming that contexts are responsive during the process. When it terminates it will either remove knowledge contradicted by these proposals, or accept and perform them, or reject them:

- c_i : PROPOSE (add_ontorel(m, c_j, d_j))
- c_i : PROPOSE (add_mapping(m, c_i, c_k))

in finite time, in accordance to the criteria in theorem 2 and maintain the defined consistency constraints. It will only reject the proposals if there is a violation of the consistency constraints.

A comment: "The consistency constraints" refers to maintaining the correct amount of consistency. If the special IS/IS2 construct (see §8.2.8) is not used, then the correct amount of redundancy is also maintained.

8.4.1 The case of adding an ontology relationship.

We now focus on the protocol description and the execution of the different processes.
We first focus on the case of adding an ontology relationship. Therefore, we must recall rule 8 in §5.4 and rule 9 in §5.5. Then we simply graphically visualise the different states and how the execution of them could progress (see Figure 50). The root of this tree starts with the proposal by context c_j to add a mapping. In all situation where there is an "OR" only one of the paths are executed, and the link described as "LOOP" can be repeated many times – but a finite amount of times. By looking at the structure and nodes of this tree marked as

bold we see that no matter what happens, then either

- Existing contradicted knowledge will be removed in several spheres of consistency (and the proposed ontology relationship will not be added).
- Nothing is done
- The proposed ontology relationship is added

In the last case this activates a request to add an ontology mapping, that could then activate the next process model (see §8.4.2), but because we will see that even that second model terminates in finite, even the process of adding of an ontology relationship terminates in finite time.

The children of the root all do some form of reasoning, but according to Theorem 3 that will happen in finite time. The grandchildren of the root send information to the contexts that then have to respond, so we have here assumed that they actually will respond in finite time.

Finally, the case when an ontology relation is added, can only happen if there is redundancy that is not dangerous (as described in §8.2.6) or neither redundancy or contradiction. So by following this process model, the infrastructure mechanism will maintain the promised consistency constraint (again, see the comment at the end of §8.4).

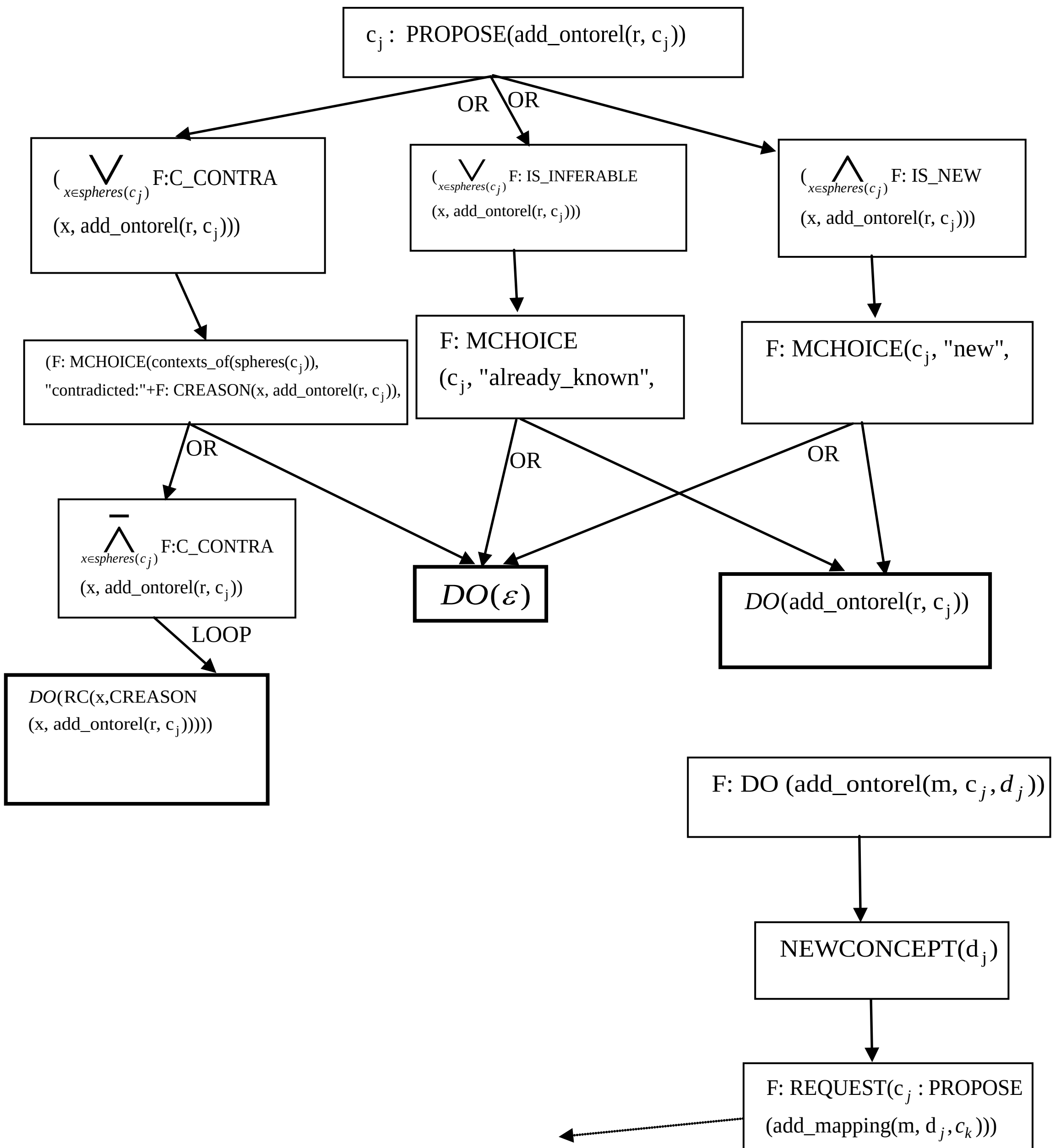
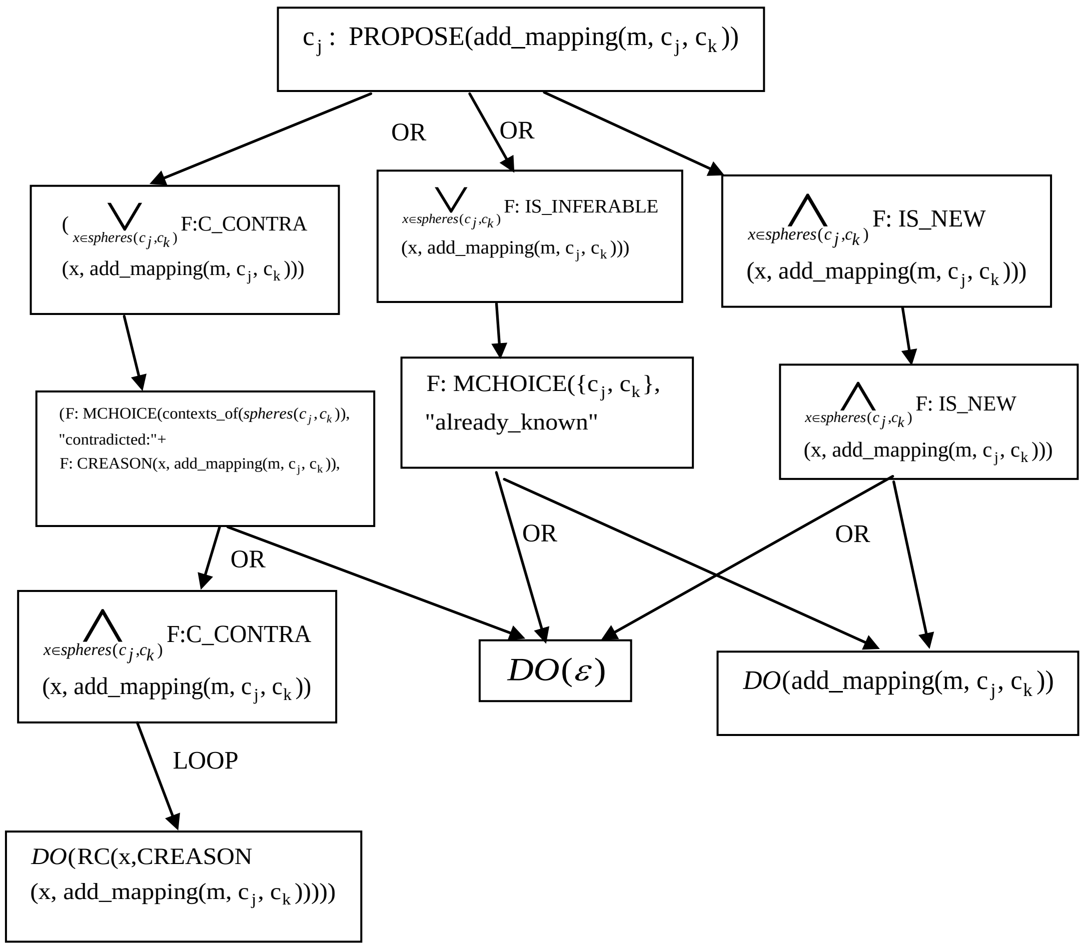

c_j : PROPOSE(add_ontorel(r, c_j))
OR
OR
($\bigvee_{x\in spheres(c_j)}$ F:C_CONTRA (x, add_ontorel(r, c_j)))
($\bigvee_{x\in spheres(c_j)}$ F: IS_INFERABLE (x, add_ontorel(r, c_j)))
($\bigwedge_{x\in spheres(c_j)}$ F: IS_NEW (x, add_ontorel(r, c_j)))
(F: MCHOICE(contexts_of(spheres(c_j)), "contradicted:"+F: CREASON(x, add_ontorel(r, c_j)),
F: MCHOICE (c_j, "already_known",
F: MCHOICE(c_j, "new",
OR
OR
OR
$\overline{\bigwedge}_{x\in spheres(c_j)}$ F:C_CONTRA (x, add_ontorel(r, c_j))
$DO(\varepsilon)$
DO(add_ontorel(r, c_j))
LOOP
DO(RC(x,CREASON (x, add_ontorel(r, c_j)))))
F: DO (add_ontorel(m, c_j, d_j))
NEWCONCEPT(d_j)
F: REQUEST(c_j : PROPOSE (add_mapping(m, d_j, c_k)))

Figure 50. A visualisation of rule 8 in §5.4 and rule 9 in §5.5 where every box corresponds to a process state. The dotted line connects to the root of figure 49.



c_j : PROPOSE(add_mapping(m, c_j, c_k))
OR
OR
($\bigvee_{x \in spheres(c_j,c_k)}$ F:C_CONTRA (x, add_mapping(m, c_j, c_k)))
$\bigvee_{x \in spheres(c_j,c_k)}$ F: IS_INFERABLE (x, add_mapping(m, c_j, c_k)))
$\bigwedge_{x \in spheres(c_j,c_k)}$ F: IS_NEW (x, add_mapping(m, c_j, c_k)))
(F: MCHOICE(contexts_of($spheres(c_j,c_k)$), "contradicted:"+ F: CREASON(x, add_mapping(m, c_j, c_k)),
F: MCHOICE({c_j, c_k}, "already_known"
$\bigwedge_{x \in spheres(c_j,c_k)}$ F: IS_NEW (x, add_mapping(m, c_j, c_k)))
OR
OR
OR
$\bigwedge_{x \in spheres(c_j,c_k)}$ F:C_CONTRA (x, add_mapping(m, c_j, c_k))
$DO(\varepsilon)$
DO(add_mapping(m, c_j, c_k))
LOOP
DO(RC(x,CREASON (x, add_mapping(m, c_j, c_k)))))

Figure 51. A visualisation of rule 7 in §5.4 where every box corresponds to a process state.

8.4.2 The case of adding an ontology mapping.

In this case a context c_j is proposing to add a mapping from its ontology to an ontology of context c_k. As before, the children of root node do reasoning in finite time and the grandchildren of the root node need to receive responses from one or several contexts and it is assumed they will do so in finite time. As before, these are the three things that can happen:

- Existing contradicted knowledge will be removed in several spheres of consistency (and the proposed ontology relationship will not be added).
- Nothing is done
- The proposed ontology relationship is added

We see that no matter which path is taken, the processing will happen in finite time.
Finally, the action to add a mappings is only performed when the proposed mappings neither creates a redundancy or contradiction, or creates a allowed redundancy. It will therefore maintain the defined consistency constraints (but see the comment at the end of §8.4).

Chapter 9 Conclusions and Future Work

9.1 Conclusions

We have defined an infrastructure mechanism that explicitly models

1) the distributed process of initiating change
2) how a context proposes to evolve its ontology
3) how mappings are proposed to evolve
4) a layered system that processes these proposal transactions by following explicit policies, and does automated reasoning (that calculates if a proposed change would introduce inconsistency or redundancy)
5) two kinds of consistency constraints that the mechanism guarantees to satisfy.

The consistency constraints are formalised using so called *spheres of consistency* that define

1) knowledge regions within which consistency is maintained and
2) a variable degree of proof-bounded consistency verification within these regions.

Our infrastructure formalism defines a protocol and its computational semantics, as well as a model theory and proof theory for the reasoning layer of the mechanism.

In our work we have proved the formal properties of correctness and completeness of the reasoning and termination of the processing for our infrastructure mechanism. Then we have evaluated the infrastructure mechanism experimentally by doing simulations using an implemented prototype. Our experiments show for example that if we bound the reasoning enough a phase transition can occur in this kind of system, beyond which constant-time and constant-memory complexity is approached. We also evaluated the usefulness of pair-wise consistency where pairs of contexts are connected by mappings and a minimal sphere of consistency (being the first approach). This is compared to the second approach, where global consistency is enforced. The evaluation shows that the first approach reduces both the amount of and the variability of the computational effort needed for processing a transaction, compared to the second approach. The benefit of the first approach over the second one increases when: the size difference increases between large core ontologies and smaller interface ontologies, and when a more complex topology of connected ontologies is used.

The new paradigm we propose is to use an infrastructure mechanism that processes ontology change proposals from autonomous entities while maintaining user-defined consistency between the ontologies of these entities. This enables semantic autonomy and decentralised control while at the same time maintaining interoperability between the entities. The conclusion of this thesis is that this new paradigm is possible and beneficial, assuming that the knowledge representation is kept simple, the ontology evolution operations are kept simple and one proposal is processed at a time. Future research should investigate if and how these results can be extended to more expressive logics, more complex ontology evolution operations, fully concurrent processing of change proposals and several instances of co-operating knowledge infrastructure mechanisms.

9.2 Future Work

Despite that the publication date was a few years ago, the publication by (Maedche et al., 2003) mentions an interesting future goal, that still has not been achieved in 2009. They describe three different parameters of an information system: information vs. activity (e.g. a web service), centralised vs. ad-hoc (i.e. P2P), and implicit vs. explicit semantic descriptions. Using these parameters they describe a vision of an "ultimate" information processing system that they call "P2P-based semantic web services". Such system uses formal semantics, manages services (web services) but is decentralised using P2P.

Our infrastructure mechanism does not manage semantics of web services, but otherwise our vision of the future is similar in so far that then systems must support concurrent processing, several infrastructure mechanisms (without any of them being the "main" one) and still maintain semantic meaning in a precise way and mappings between different semantics.

It is not obvious that such a system can be built, but we nevertheless now present a list of improvements and generalisations of the current infrastructure mechanism that should be attempted in the future.

9.2.1 Support other logics for knowledge representation

In this infrastructure mechanism we have used a simple logic for the ontologies and somewhat more complex logic for the ontology mappings. In the future, it would be interesting to investigate if the system could use some other logics for ontologies, e.g. a modest subset of OWL. One would then have to define incremental bounded consistency for that logic, which might not be easy. However, if it is possible then it would be interesting to investigate if proof-bounded consistency would improve the worst-case time-complexity.

9.2.2 Model API between infrastructure mechanism and semantic applications

As was motivated in chapter 1, the purpose of a knowledge infrastructure is to be used by semantic applications that utilize the ontologies. The PROPOSE() statement could be a part of an API that semantic applications can call when they need to change on ontology or mapping, and the infrastructure would, after receiving and processing it, either perform the proposed change using DO() or do nothing, but it should then send a message back to the semantic application notifying it of the change it has done. As was motivated in chapter 1, then the semantic application does not have to focus on evolving ontologies or maintaining consistency and interoperability because that work has been *delegated* to the infrastructure mechanism that performs it as a service on behalf of one or (more likely) several semantic applications.

When designing a protocol that is the core of the infrastructure mechanism, one should then distinguish between the statements that are a part of the API and that can be seen and called from *outside* and the statements that are only used *inside* the infrastructure mechanism. So future work should focus on this distinction and on adding more statements to the API that can be accessed from outside. E.g. statements for creating and removing instances, or statements for notifying that an ontological concept is *used* by a semantic application (e.g. it contains stored knowledge that uses the concept). These statements could be inspired by the locking mechanism in databases where there are several read and write statements that have different levels of exclusivity.

9.2.3 Concurrency

Currently, the infrastructure is processing single proposal at a time, whereas future research should investigate if the infrastructure mechanism could process several proposals concurrently because they are created concurrently.

9.2.4 Expansion and evolution of spheres to new contexts

Currently, the spheres of consistency are defined to contain certain ontologies and mapping sets and these evolve, so the knowledge contents of the sphere change. However, in the future it should also be investigated what happens if completely new ontologies are added to an existing spheres of consistency and new sets of mappings. In that situation the knowledge wouldn't evolve but the sphere of consistency would evolve and change the territory that it is encompassing. That is an interesting functionality because then the infrastructure mechanism can be used on a small scale first and e.g. only support two ontologies but then gradually expend to include more and more ontologies and their mappings, so that its organizing ability would gradually include more and more ontologies.

9.2.5 Supporting other kinds of consistency

In §2.1.2.1 we described the difference between logical inconsistency and logical "incoherence" as it is defined by some researchers. If that form of incoherence is to be disallowed then a certain change is needed in the reasoning layer. The modified reasoning layer would investigate if a certain combination of relationships requires some concept models to be empty – i.e. there can't be any instances of these concepts. That modified reasoner would disallow such situations that required empty models, because if an instance is created of such a concept then that it leads to a conventional inconsistency. Therefore, that reasoner would only accept changes (without classifying them as creating an inconsistency) that do not lead to any concept having an empty model.

9.2.6 Consistency and/or domain-dependent constraints

Logical inconsistency is only one form of inconsistency. In some application, e.g. healthcare informatics, then it might make sense to focus even more on so called *domain-dependent consistency* or *user-defined consistency*. These inconsistency models define certain constraints that cannot be violated. E.g. they could model that a patient that has a certain illness cannot be prescribed a certain medication (due to an interaction), or computer that is configured in a certain way cannot be fitted with an extra component (due to a technical conflict).

9.2.7 Instance support

There is disagreement between researchers if instances should be seen as part of the ontologies or be seen as something separate. In this thesis we have focused on changes at the conceptual level, and due to limited time, we have not included processes that manage instances, e.g. that add or delete instances. If instance support is added, then the reasoning system would have to propagate the influence an instance through the ontological relationships and see if that leads to conflicts with other instances. Also, currently the system maintains logical consistency, and in some cases this would require certain concepts to have empty models. If an instance is created for such a concept, then that would also lead to a contradiction.

9.2.8 Supporting several infrastructure mechanisms and their integration

During our evaluation (see Chapter 7) we have investigated a single infrastructure mechanism. If we look closely at some the evaluation scenarios we will see that they model two or three organizations having several ontologies each. To simplify things, we assumed during the evaluation that all these organizations and their divisions (that correspond to a context and ontology) are being managed by a single infrastructure mechanism, that accepts a proposal at a time. In the future, these assumptions should be relaxed and every organization could then run its own infrastructure mechanism instance, and the infrastructure mechanisms must then support a new functionality of being able to negotiate with each other about ontologies that are connected with mappings but that belong to two different infrastructure mechanism. This case becomes more complicated, because the infrastructure mechanisms must then also be able to support distributed reasoning that starts in infrastructure mechanism and might continue in the reasoner of another infrastructure mechanism.

Such a generalised setting is more complex, but it would have very interesting applications because it would get a step closer to a general purpose technology for maintaining interoperability between evolving ontologies. The reason for this is that the infrastructure mechanism is characterised by a formal specification that could be implemented by different software vendors or even service providers. This approach is more realistic, because it supports pluralism at the software/server level (and does not require that a single software provider has monopoly on "the perfect" software for providing interoperability) while still marinating a clear structure at the process and semantic level – i.e. it still would clearly define the process for maintaining consistency and interoperability.

References

Antoniou, G., van Harmelen, F., "A Semantic Web Primer (second edition)", The MIT press, 2008.

Bell, D., Qi, G, Liu, W. "Approaches to Inconsistency Handling in Description-Logic Based Ontologies", In: *OTM Confederated International Workshops and Posters*, pp. 1303-1311. Springer, Vilamoura, Portugal, November 2007.

Benerecetti, M., Bouquet, P., and Ghidini, C. Contextual reasoning distilled. Philosophical Foundations of Artificial Intelligence, 12(3), July 2000.

Bennet, D., Bennet, A., "The Rise of the Knowledge Organization", In: *Handbook on Knowledge Management, vol.1*, C.W. Holsapple (Ed.), pages 5-20. 2003.

Bonifacio, M., Bouquet, P., and Manzardo, A. "A distributed intelligence paradigm for knowledge management". In: *AAAI Spring Symposium Series 2000 on Bringing Knowledge to Business Processes*. AAAI, 2000.

Bonifacio, M., Cuel, R., Mameli, G., Nori, M., "A Peer-to-Peer Architecture for Distributed Knowledge Management", In: *Proceedings of the 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB'2002)*, 2002.

Bouquet, P., Ghidini, C., Giunchiglia, F., Blanzieri, E., "Theories and uses of context in knowledge representation and reasoning", IRST Technical Report 0110-28, Istituto Trentino di Cultura, October 2001.

Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies, Proceedings of the Second International Semantic Web Conference, K. Sekara and J. Mylopoulis (Ed.), pages 164-179, Lecture Notes in Computer Science. Springer Verlag. 2003.

Bouquet, P., Serafini, L., Zanobini, S., "Semantic Coordination: A New Approach and an Application", *International Semantic Web Conference 2003*, pages 130-145. 2003b.

Bouquet, P., Giunchiglia, F., "A Context-Based Framework for Mental Representation", In: *20th annual conference of the cognitive science society (CogSci'98)* , Gernsbacher, M. A., Derry S. J. (ed.), pp. 392-397, University of Wisconsin-Madison, August 1998.

Brachman, R., Levesque, H., Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence), 2004.

Broekstra, J., Ehrig, M., Haase, P.,van Harmelen, F., Kampman, A., Sabou, M., Siebes, R., Staab, S., Stuckenschmidt, H., Tempich, Ch., "A metadata model for Semantics-Based Peer-to-Peer Systems", In: *Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing,* 2003.

Bussler, C., “Business-to-Business Integration needs a Meta Business Enterprise Ontology”, In: *Proceedings of the International Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations (OES-SEO2001).* Rome, Italy, September 2001.

Desouza, K.C., Evaristo, J.R. “Managing Knowledge in Distributed Projects” *Communications of the ACM,* Volume 47, Issue 4. Pages: 87 – 91. April 2004.

Euzenat, J., “Corporate Memory through Cooperative Creation of Knowledge Base Systems and Hyper-Documents”. In: *Proc. of Knowledge Acquisition Workshop (KAW'96),* Banff, Canada, 1996.

Everett, J. O., Bobrow, D. G., Polanyi, L., van den Berg, M. H., Condoravdi, C., Crouch, R., Stolle, R., di Paiva, V., “Making Ontologies Work for Resolving Redundancies Across Documents”, In: *Communication of the ACM,* Volume 45 Number 2. pages 55-60., February, 2002.

Froehner, T., Nickles, M. & Weiß, G.: Towards modeling the social layer of emergent knowledge using open ontologies., In: *ECAI Workshop on Agent-Mediated Knowledge Management (AMKM, Workshop Notes pp. 10-19).* (2004).

Gamut, L. T. F. “Logic, Language, and Meaning, Volume 2: Intensional Logic and Logical Grammar”, 1991.

Ghidini, C., Giunchiglia F., “Local Model Semantics, or Contextual Reasoning = Locality + Compatibility”, In: *Artificial Intelligence*, 127(2), pages 221-259, 2001.

Giunchiglia, F., “Contextual reasoning”, In: *Epistemologia - Special Issue on I Linguaggi e le Macchine,* XVI, pages 345-364, 1993.

Giunchiglia, F., Shvaiko, P., “Semantic matching”, *Knowl. Eng. Rev.,* vol.18, nr. 3, pages 265-280. Cambridge University Press. 2003.

Guha, R., McCarthy, J., “Varieties of contexts”, In: *4th International and Interdisciplinary Conference*, *CONTEXT 2003*, LNAI 2680, pages 164-177. Springer-Verlag, January 2003.

Halladay, S. M., Milligan, C. A., "The Application of Network Science Principles to Knowledge Simulation", In: *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 8 - Volume 8*. Page: 80243.3. 2004.

Haase, P., Stojanovic, L., “Consistent Evolution of OWL Ontologies”, In: *Proceedings of the Second European Semantic Web Conference,* Heraklion, Greece, 2005, volume 3532 of Lecture Notes in Computer Science, pp. 182-197. Gómez-Pérez, A., Euzenat, J. (Eds.) Springer, May 2005.

Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y., “A Framework for Handling Inconsistency in Changing Ontologies”, In: *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)* (Eds. Y. Gil, E. Motta, V. R. Benjamins, M. A. Musen), volume 3729 of LNCS, pp. 353-367. Springer, November 2005.

Heflin, J., Hendler, J., “Dynamic Ontologies on the Web”, In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).* AAAI/MIT Press, Menlo Park, CA, 2000.

Horrocks, I. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

Horrocks, I. “Reasoning with expressive description logics: Theory and practice", In, *Proceedings of the 18th International Conference* on *Automated Deduction (CADE 2002),* A. Voronkov (ed.). 2002.

Huang, Z., Harmelen, F. v., Teije, A. t. "Reasoning with Inconsistent Ontologies". In: IJCAI 2005: pages 454-459. 2005.

Kalfoglou, Y., Schorlemmer, M., “Ontology mapping: the state of the art.”, *The Knowledge Engineering Review* 18(1):pp. 1-31. 2003.

Kingston, J., Macintosh, A., “Knowledge management through multi-perspective modelling: representing and distributing organizational memory”, In: *Knowledge-Based Systems*, Vol. 13, Issues 2-3, pages 121-131, April 2000.

de Kleer, J. "An assumption-based TMS", In: *Artificial Intelligence* 28 (2), pages 127-162. 1986.

Ma, Y., Qi, G., Hitzler, P., Lin, Z.: An Algorithm for Computing Inconsistency Measurement by Paraconsistent Semantics. In: Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'07), (2007).

Ma, Y., Qi, G., Hitzler, P., Lin. Z., "Measuring Inconsistency for Description Logics Based on Paraconsistent Semantics", In: *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertaint*. Lecture Notes In Artificial Intelligence; Vol. 4724. Pages: 30 – 41. 2007b.

Ma. Y., Hitzler, P., Lin, Z., "Algorithms for Paraconsistent Reasoning with OWL". In: *Proceedings of the 4th European conference on The Semantic Web: Research and Applications.* Lecture Notes In Computer Science; Vol. 4519. Pages: 399 – 413. 2007c.

Maedche, A., Motik, B., Silva, N., Volz, R., "MAFRA - A MApping FRAmework for Distributed Ontologies", In: *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web.* Lecture Notes In Computer Science; Vol. 2473, pages 235 – 250, 2002.

Maedche, A., Motik, B., Stojanovic, L., Studer, R., and Volz, R., “An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies,” In: *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 20–24, 2003, ACM, pages 439–448, New York, 2003.

Maedche, A., Motik, B., Stojanovic, L. "Managing multiple and distributed ontologies on the Semantic Web", In: *The VLDB Journal — The International Journal on Very Large Data Bases.* Volume 12 , Issue 4. Pages: 286 – 302. 2003b.

Maedche, A., Staab, S., “Services on the Move - Towards P2P-Enabled Semantic Web Services”, *Proceedings of the Tenth International Conference on Information Technology and Travel & Tourism ENTER 2003*, Helsinki, January 2003c.

Maedche, A., Motik, B., Stojanovic, L., Studer, R., and Volz, R., “Ontologies for Enterprise Knowledge Management”, In: *IEEE Intelligent Systems*, Volume 18, Number 2, pages 26-33, March/April 2003d.

Majkic, Z., “Intensional Logic and Epistemic Independency of Intelligent Database Agents”, 2nd International Workshop on Philosophy and Informatics (WSPI 2005), Kaiserslautern, Germany, April 11-13, 2005.

McNeill, F., Bundy, A., Walton, C., “Planning from rich ontologies through translation between representations”, technical report EDI-INF-RR-0244, Edinburgh University, School of Informatics, June 2005.

Noy, N. F., and McGuinness, D. L. ``Ontology Development 101: A Guide to Creating Your First Ontology". Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

Noy, M., Klein, M., "Ontology Evolution: Not the Same as Schema Evolution", *Knowledge and Information Systems*, vol. 6, pages 428-440, 2004.

Parsia, B. and Patel-Schneider, P. F., "Meaning and the semantic web", In: *Proceedings of Identity, Reference, and the Web (IRW2006) Workshop, at the WWW2006 Conference*, 2006.

Reimer, U., "Knowledge integration for building organization memories", In: *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Alberta, Canada, 1998.

Robertson, D.: Multi-agent Coordination as Distributed Logic Programming, In: Lecture Notes in Computer Science, Volume 3132, Pages 416 – 430, (2004)

Serafini, L., Tamilin, A., “DRAGO: Distributed Reasoning Architecture for the Semantic Web”. In: Proc. of the Second European Semantic Web Conference (ESWC'05), 2005.

Staab, S., Studer, R., Schnurr, H.-P., Sure, Y., "Knowledge processes and ontologies", In: *IEEE Intelligent Systems*, 16(1):26–34, 2001.

Stojanovic, L., Maedche, A., Motik, B., "User-Driven Ontology Evolution Management", In: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web(Lecture Notes in Computer Science)*, volume 2473/2002, pages 133-140, 2002.

Stojanovic, L., Schneider, J., Maedche, A., Libischer, S., Studer, R., Lumpp, Th., Abecker, A., Breiter, G., Dinger, J., “The role of ontologies in autonomic computing systems”, In: *IBM Systems Journal*, v.43 n.3, p.598-616, July 2004

Stuckenschmidt, H., Timm, I. J., “Adapting Communication Vocabularies Using Shared Ontologies” In: *Proceedings of the Second International Workshop on Ontologies in Agent Systems, Workshop at 1st International Conference on Autonomous Agents and Multi-Agent Systems,* Cranefield, S. et al. (Eds.), pages 6-12, Bologna, Italy, 2002.

Wang, Y., Bao, J., Haase, P., Qi, G.: Evaluating Formalisms for Modular Ontologies in Distributed Information Systems. In: Proceedings of The First International Conference on Web Reasoning and Rule Systems (RR2007), pp. 178-182. Springer, Innsbruck, Austria, June (2007).

Zhao Y., Wang K., Topor R., Pan J. Z. and Giunchiglia F., “Semantic Cooperation and Knowledge Reuse by Using Autonomous Ontologies”, In: *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, pages 666-679, Busan, Korea, 2007.

Zurawski, M.: Towards a context-sensitive distributed knowledge management system for the knowledge organization, In: Workshop on Knowledge Management and the Semantic Web, 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004), UK, (2004)

Zurawski, M.: Reasoning about multi-contextual ontology evolution, In: The First International Workshop on Context and Ontologies: Theories, Practice and Applications, The Twentieth National Conference on Artificial Intelligence (AAAI-05), July 9-13, Pittsburgh, PA, USA, (2005).

Zurawski, M.: "Distributed Multi-Contextual Ontology Evolution - A Step Towards Semantic Autonomy", In: EKAW 2006 - 15th International Conference on Knowledge Engineering and Knowledge Management, 2nd-6th October 2006, Podebrady, Czech Republic, (2006)

Zurawski, M., Smaill, A., Robertson, D., "Bounded Ontological Consistency for Scalable Dynamic Knowledge Infrastructures", *ASWC08 – 3rd Asian Semantic Web Conference*, 8th-11th December, Bangkok, Thailand, 2008.

Appendix A

A.1.1 Proof of four cases when R= Cor(A$_i$, B$_j$)

Proof for the four cases when R= Cor(A$_i$, B$_j$) and R'= Cor (B$_j$, C$_k$) | Is (B$_j$, C$_k$) | Is2 (B$_j$, C$_k$) | DISJOINT (B$_j$, C$_k$).

Let us assume that R= Cor(A$_i$, B$_j$), and

R'= Cor (B$_j$, C$_k$) | Is (B$_j$, C$_k$) | Is2 (B$_j$, C$_k$) | DISJOINT (B$_j$, C$_k$)

We then have

$$R = \boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j))))$$

and

$$R' = \boldsymbol{N}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_j(z_j))))$$

where

$\Theta(B_j(x_j), C_j(y_j))$ is one of four different Boolean functions.

We then have

$$R \wedge R' \Rightarrow$$

$$\boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{N}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))) \Rightarrow$$

$$V_{(i,j),s}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))\big|_{z=s}] = 1 \wedge$$
$$\forall(st \in S)(L(s, st) \to V_{(i,j),st}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))\big|_{z=st}] = 1) \wedge$$
$$V_{(i,j),s'}[\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))\big|_{z=s'}] = 1 \wedge$$
$$\forall(st \in S)(L(s', st) \to V_{(i,j),st}[\forall y_j, z_k(\mathrm{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))\big|_{z=st}] = 1) \Rightarrow$$

$$V_{(i,j),s'}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j))) \wedge$$
$$\forall y_j, z_k(\text{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))\big|_{z=s'}] = 1 \wedge$$
$$\forall(st \in S)(L(s', st) \to V_{(i,j),st}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j))) \wedge$$
$$\forall y_j, z_k(\text{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))\big|_{z=st}] = 1) \Rightarrow$$

$$V_{(i,j),s'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to \Theta(A_i(x_i), C_k(z_k)))\big|_{z=s'}] = 1 \wedge$$
$$\forall(st \in S)(L(s', st) \to V_{(i,j),st}[\forall x_i, z_k(\text{Rel}(x_i, y_j) \to \Theta(A_i(x_i), C_k(z_k)))\big|_{z=st}] = 1) \Rightarrow$$

$$\boldsymbol{N}_{s'}(\forall x_i, z_k(\text{Rel}(x_i, z_k) \to \Theta(A_i(x_i), C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall x_i, z_k(\text{Rel}(x_i, z_k) \to \Theta(A_i(x_i), C_k(z_k))))$$

$$= R''$$

By looking at this derivation we learn that R'' is the same relationship as R' with the exception that R' operates on the arguments $(B_j(y_j), C_k(z_k))$ whereas R'' operates on the arguments $(A_i(x_i), C_k(z_k))$. This holds if we assume that R operates on the variables $(A_i(x_i), B_j(y_j))$.

Therefore, if R= Cor(A$_i$, B$_j$), and
R'= Cor (B$_j$, C$_k$) | Is (B$_j$, C$_k$) | Is2 (B$_j$, C$_k$) | DISJOINT (B$_j$, C$_k$)
then
R''= Cor (A$_i$, C$_k$) | Is (A$_i$, C$_k$) | Is2 (A$_i$, C$_k$) | DISJOINT (A$_i$, C$_k$)

So we have verified the correctness of these four cases, and now only the completeness remains.

For two of these cases (when R'= Is (B$_j$, C$_k$) | Is2 (B$_j$, C$_k$)) we have to prove that we couldn't have inferred anything more general, i.e. that it is not possible to infer that R''= Cor (A$_i$, C$_k$) for both of these cases. We do this by showing a counter-example.

Let R= Cor(A$_i$, B$_j$) and R'= Is (B$_j$, C$_k$) | Is2 (B$_j$, C$_k$). We assume that R''= Cor (A$_i$, C$_k$) | Cor (A$_i$, C$_k$).
We have to investigate if

$R \wedge R' \Rightarrow R''$

This statement is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I(A_1(a_1))$=false	$I(B_2(b_2))$=false	$I(C_3(c_3))$=true
$I_{(1,2)}(\text{Rel})=\{(a_1, b_2)\}$	$I_{(1,3)}(\text{Rel})=\{(a_1, c_3)\}$	$I_{(2,3)\,)}(\text{Rel})=\{(b_2, c_3)\}$

Therefore, our initial assumption that R"= Cor (A_i, C_k) | Cor (A_i, C_k) is therefore false.

A.1.2 Proof of four cases when R´= Cor(B_j, C_k) and R≠ COMPATIBLE(A_i, B_j)

Proof for the four cases when R= Cor(A_i, B_j) | Is(A_i, B_j) | Is2(A_i, B_j) | DISJOINT(A_i, B_j) and R'= Cor(B_j, C_k).

Let us assume that R= Cor(A_i, B_j) | Is(A_i, B_j) | Is2(A_i, B_j) | DISJOINT(A_i, B_j)
and R'= Cor(B_j, C_k).

We then have

$$R = \boldsymbol{N}_s(\forall x_i y_j(\text{Rel}(x_i, y_j) \rightarrow \Theta(A_i(x_i), B_j(y_j)))) \wedge \\ \boldsymbol{G}_s(\forall x_i y_j(\text{Rel}(x_i, y_j) \rightarrow \Theta(A_i(x_i), B_j(y_j))))$$

and

$$R = \boldsymbol{N}_{s'}(\forall y_j z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))) \wedge \\ \boldsymbol{G}_{s'}(\forall y_j z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k))))$$

where

$\Theta(B_j(x_j), C_j(y_j))$ is one of four different Boolean functions.

We then have

$R \wedge R' \Rightarrow$
$\boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to \Theta(A_i(x_i), B_j(y_j)))) \wedge$
$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to \Theta(A_i(x_i), B_j(y_j)))) \wedge$
$\boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (B_j(y_j) \leftrightarrow C_k(z_k)))) \wedge$
$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (B_j(y_j) \leftrightarrow C_k(z_k))))$
$\Rightarrow$
$\boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to \Theta'(B_j(y_j), A_i(x_i)))) \wedge$
$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to \Theta'(B_j(y_j), A_i(x_i)))) \wedge$
$\boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (C_k(z_k) \leftrightarrow B_j(y_j)))) \wedge$
$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (C_k(z_k) \leftrightarrow B_j(y_j))))$
$\Rightarrow$

$\boldsymbol{N}_{s'}(\forall z_k, y_j(\text{Rel}(y_j, z_k) \to (C_k(z_k) \leftrightarrow B_j(y_j)))) \wedge$
$\boldsymbol{G}_{s'}(\forall z_k, y_j(\text{Rel}(y_j, z_k) \to (C_k(z_k) \leftrightarrow B_j(y_j))))$
$\boldsymbol{N}_s(\forall y_j, x_i(\text{Rel}(x_i, y_j) \to \Theta'(B_j(y_j), A_i(x_i)))) \wedge$
$\boldsymbol{G}_s(\forall y_j, x_i(\text{Rel}(x_i, y_j) \to \Theta'(B_j(y_j), A_i(x_i)))) \wedge$
$\Rightarrow$ [substitution of variable names]
$\boldsymbol{N}_s(\forall y_j, x_i(\text{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$
$\boldsymbol{G}_s(\forall y_j, x_i(\text{Rel}(x_i, y_j) \to (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$
$\boldsymbol{N}_{s'}(\forall z_k, y_j(\text{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k)))) \wedge$
$\boldsymbol{G}_{s'}(\forall z_k, y_j(\text{Rel}(y_j, z_k) \to \Theta(B_j(y_j), C_k(z_k))))$

By using the symmetry of equivalence and the Boolean functions, we have now reached the same expression as the start of the corresponding calculation in §A.1.1 Proof of four cases when R= Cor(A_i, B_j).
Therefore, also here
R"= Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k)

A.1.3 The case when R= Cor(A_i, B_j) and R'=Compatible(B_j, C_k)

Let us assume that R= Cor(A_i, B_j), and
R'= COMPATIBLE (B_j, C_k)

We then have

$$R = \boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j))))$$

and

$$R' = \boldsymbol{F}_{s'}(\exists y_j, z_k(\mathrm{Rel}(y_j, z_k) \wedge B_j(y_j) \wedge C_k(z_k)))$$

$$R \wedge R' \Rightarrow$$

$$\boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))) \wedge$$
$$\boldsymbol{F}_{s'}(\exists y_j, z_k(\mathrm{Rel}(y_j, z_k) \wedge B_j(y_j) \wedge C_k(z_k)))$$

$$V_{(i,j),s}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))\big|_{z'=s}] = 1 \wedge$$
$$\forall(st \in S)(L(s, st) \rightarrow V_{(i,j),st}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))\big|_{z'=st}] = 1) \wedge$$
$$\exists(st' \in S)(L(s'-1), st') \wedge V_{(i,j),st'}[\exists y_j, z_k(\mathrm{Rel}(y_j, z_k) \wedge B_j(y_j) \wedge C_k(z_k))\big|_{z'=st'}]$$

$$\forall(st \in S)(L(s-1, st) \rightarrow V_{(i,j),st}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \rightarrow (A_i(x_i) \leftrightarrow B_j(y_j)))\big|_{z'=st}] = 1) \wedge$$
$$\exists(st' \in S)(L(s'-1), st') \wedge V_{(i,j),st'}[\exists y_j, z_k(\mathrm{Rel}(y_j, z_k) \wedge B_j(y_j) \wedge C_k(z_k))\big|_{z'=st'}]$$

$$\exists(st' \in S)(L(s'-1), st') \wedge V_{(i,j),st'}[\exists x_i, z_k(\mathrm{Rel}(x_i, z_k) \wedge A_i(x_i) \wedge C_k(z_k))\big|_{z'=st'}] =$$

COMPATIBLE (A_i, C_k)

=R"

In §A.2.2 If L⊨ COMPATIBLE (A_i, C_k), then L does not entail other relations.we show that if R ∧R' ⊨ COMPATIBLE (A_i, C_k) and only one relation holds between A_i and C_k then R ∧R' does not entail any other relation. We have therefore verified that if

R= Cor(A_i, B_j), and R'= COMPATIBLE (B_j, C_k)

then R ∧R' ⇒ COMPATIBLE (A_i, C_k)

A.1.4 The case when R=Compatible(A_i, B_j) and R'= Cor(B_j, C_k)

Let us assume that R= COMPATIBLE (A_i, B_j), and

R'= COR(B_j, C_k)

We then have

$$R = \boldsymbol{F_r}(\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j)))$$

and

$$R = \boldsymbol{N}_s(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_s(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k))))$$

$$R \wedge R' \Rightarrow$$

$$\boldsymbol{F}_s(\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))) \wedge$$
$$\boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))) \Rightarrow$$

$$\exists(st \in S)(L(s-1), st) \wedge V_{(i,j),st}[\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st}] \wedge$$
$$V_{(i,j),s'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \rightarrow V_{(i,j),st'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))\big|_{z'=st'}] = 1)$$
$$\Rightarrow$$

$$\exists(st \in S)(L(s-1), st) \wedge V_{(i,j),st}[\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st}]$$
$$\forall(st' \in S)(L(s'-1, st') \rightarrow V_{(i,j),st'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (B_j(y_j) \leftrightarrow C_k(z_k)))\big|_{z'=st'}] = 1)$$
$$\Rightarrow \text{(assuming } s > s')$$

$$\exists(st \in S)(L(s-1), st) \wedge V_{(i,j),st}[\exists x_i, z_k(\text{Rel}(x_i, z_k) \wedge A_i(x_i) \wedge C_k(z_k))\big|_{z'=st}]$$

In §A.2.2 If L⊨ COMPATIBLE (A_i, C_k), then L does not entail other relations. we show that if R $\wedge$R' $\rightarrow$ COMPATIBLE (A_i, C_k) and only one relation holds between A_i and C_k then R $\wedge$R' does not imply any other relation. We have therefore verified that if

R= COMPATIBLE (A_i, B_j), and R'= COR(B_j, C_k)

then R $\wedge$R' $\Rightarrow$ COMPATIBLE (A_i, C_k)

A.1.5 The case when R=Is(A_i, B_j) and R'= Is(B_j, C_k)

Let us assume that R=IS(A_i, B_j) and R'=IS(B_j, C_k)

We then have

$$R = \boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j))))$$

and

$$R' = \boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (\neg B_j(y_j) \vee C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (\neg B_j(y_j) \vee C_k(z_k))))$$

$$R \wedge R'$$
$$\Rightarrow$$

$$V_{(i,j),s}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j)))\big|_{z'=s}] = 1 \wedge$$
$$\forall(st \in S)(L(s, st) \to V_{(i,j),st}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j)))\big|_{z'=st}] = 1) \wedge$$
$$V_{(i,j),s'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \to (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=st'}] = 1)$$
$$\Rightarrow$$

$$V_{(i,j),s'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (\neg A_i(x_i) \vee C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (\neg A_i(x_i) \vee C_k(z_k)))\big|_{z'=st'}] = 1)$$

= IS(A_i, C_k)

We have to investigate if $R \wedge R' \to$ COR(A_i, C_k)

Therefore, given that R=IS(A_i, B_j) and R'=IS(B_j, C_k), assume that $R \wedge R' \to$ COR(A_i, C_k)

$$R \wedge R' \to COR(A_i, C_k)$$
$$\Rightarrow$$

$$V_{(i,j),s'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (\neg A_i(x_i) \vee C_k(z_k)))\big|_{z=s}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (\neg A_i(x_i) \vee C_k(z_k)))\big|_{z=st'}] = 1) \to$$
$$V_{(i,j),s'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (A_i(x_i) \leftrightarrow C_k(z_k)))\big|_{z=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall x_i, z_k(\text{Rel}(x_i, z_k) \to (A_i(x_i) \leftrightarrow C_k(z_k)))\big|_{z=st'}] = 1)$$
$$\Rightarrow$$

$$V_{(i,j),s'}[\forall x_i, z_k(\neg A_i(x_i) \vee C_k(z_k)) \to (A_i(x_i) \leftrightarrow C_k(z_k))\big|_{z=s'}]$$

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{C_2,c_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{c_2\}$
$I_{1,1}(A_1(a_1))$=false	$I_{2,1}(C_2(c_2))$=true

$I_{(1,2),1}(\text{Rel})=\{(a_1, c_2)\}$

Because we show in §A.2.3 If L$\models$ IS (A_i, C_k) and L$\not\models$COR(A_i, C_k) then L does not entail other relations. that "If L$\models$ IS (A_i, C_k) and L$\not\models$COR(A_i, C_k) then L does not imply other relations" we have now proved that:

If R=Is(A_i, B_j) and R'=Is(B_j, C_k) then

R $\wedge$ R' $\Rightarrow$ Is(A_i, C_k) and

R $\wedge$ R' does not imply other relations.

A.1.6 The case of R=Is2(A_i, B_j) and R'= Is2(B_j, C_k)

Firstly, we show that Is(A_i, B_j)=Is2(B_j, A_i)

Is(A_i, B_j)=

$$\boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg A_i(x_i) \vee B_j(y_j))))$$
$$=$$
$$\boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (B_j(y_j) \vee \neg A_i(x_i)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (B_j(y_j) \vee \neg A_i(x_i))))$$

=Is2(B_j, A_i)

Now assume that R= Is2(A_i, B_j) and R'=Is2(B_j, C_k), then

R $\wedge$ R'= Is2(A_i, B_j)$\wedge$ Is2(B_j, C_k) = Is(B_j, A_i) $\wedge$ Is(C_k, B_j) = Is(C_k, B_j) $\wedge$ Is(B_j, A_i) $\to$ Is(C_k, A_i) = Is2(A_i, C_k)

Does R $\wedge$ R' $\Rightarrow$ Cor(A_i, C_k)?

Assume so, then:

R $\wedge$ R' $\Rightarrow$ Cor(A_i, C_k)

Is2(A_i, B_j)$\wedge$ Is2(B_j, C_k) $\Rightarrow$Cor(A_i, C_k)

Is(C_k, A_i) $\Rightarrow$ Cor(A_i, C_k)

Is(C_k, A_i) $\Rightarrow$ Cor(C_k, A_i)

We have shown above that this is not true, therefore it was false to assume that

R $\wedge$ R' $\Rightarrow$ Cor(A_i, C_k).

We show in §A.2.4 If L$\models$ IS2 (A_i, C_k) and and L$\not\models$COR(A_i, C_k) then L does not entail other relations.that "A.2.4 If L$\models$ IS2 (A_i, C_k) and and L$\not\models$COR(A_i, C_k) then L does not entail other relations."

Therefore:

If R=IS2(A_i, B_j) and R'=IS2(B_j, C_k) then

R $\wedge$ R' $\Rightarrow$ IS2(A_i, C_k) and

R $\wedge$ R' does not imply other relations.

A.1.7 The case of R=COMPATIBLE(A_i, B_j) and R'= IS(B_j, C_k)

Assume that R=COMPATIBLE(A_i, B_j) and R'= IS(B_j, C_k)

We then have

$$R \wedge R'$$

$$\Rightarrow$$

$$\boldsymbol{F}_s(\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))) \wedge$$
$$\boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (\neg B_j(y_j) \vee C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (\neg B_j(y_j) \vee C_k(z_k))))$$

$$\Rightarrow$$

$$\exists(st \in S)(L(s-1), st) \wedge V_{(i,j),st}[\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st}] \wedge$$
$$V_{(i,j),s'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \rightarrow V_{(i,j),st'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \rightarrow (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=st'}] = 1)$$

$$\Rightarrow$$

$$\exists(st \in S)(L(s-1),st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i,y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st}] \wedge$$
$$\forall(st' \in S)(L(s'-1,st') \to V_{(i,j),st'}[\forall y_j, z_k(\mathrm{Rel}(y_j,z_k) \to (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=st'}] = 1)$$
$\Rightarrow$ (assuming s>s′)

$$\exists(st \in S)(L(s-1),st) \wedge V_{(i,j),st}[\exists x_i, z_k(\mathrm{Rel}(x_i,z_k) \wedge A_i(x_i) \wedge C_k(z_k))\big|_{z'=st}]$$

= COMPATIBLE(A_i, B_j)

Notice that if s<s´ then we have

$$\exists(st \in S)(L(s-1),st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i,y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st}] \wedge$$
$$\forall(st' \in S)(L(s'-1,st') \to V_{(i,j),st'}[\forall y_j, z_k(\mathrm{Rel}(y_j,z_k) \to (\neg B_j(y_j) \vee C_k(z_k)))\big|_{z'=st'}] = 1)$$

we could construct the following counter-model:

$L_{1,3}$={A_1,a_1}	$L_{2,3}$={B_2,b_2}	$L_{3,3}$={C_3,c_3}
$D_{1,3}$={a_1}	$D_{2,3}$={b_2}	$D_{3,3}$={c_3}
$I_{1,3}$(A_1(a_1))=true	$I_{2,3}$(B_1(b_1))=false	$I_{3,3}$(C_3(c_3))=true
$I_{(1,3),3}$(Rel)={(a_1, c_2)}	$I_{(1,2),3}$(Rel)={(a_1, b_2)}	$I_{(2,3),3}$(Rel)={(b_2, c_3)}

In §A.2.2 If L⊨ COMPATIBLE (A_i, C_k), then L does not entail other relations.we show that if R $\wedge$R' $\Rightarrow$COMPATIBLE (A_i, C_k) and only one relation holds between A_i and C_k then R $\wedge$ R' does not imply any other relation.

A.1.8 The case when R= DISJOINT(A_i, B_j) and R'= IS2(B_j, C_k)

Assume that R= DISJOINT(A_i, B_j) and R'= IS2(B_j, C_k).

We then have

$$R = \boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i,y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i,y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j))))$$

and

$$R' = \boldsymbol{N}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j,z_k) \to (B_j(y_j) \vee \neg C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\mathrm{Rel}(y_j,z_k) \to (B_j(y_j) \vee \neg C_j(z_j))))$$

We then have

$R \wedge R' \Rightarrow$

$$\boldsymbol{N}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j)))) \wedge$$
$$\boldsymbol{G}_s(\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j)))) \wedge$$
$$\boldsymbol{N}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to B_j(y_j) \vee \neg C_k(z_k)))) \wedge$$
$$\boldsymbol{G}_{s'}(\forall y_j, z_k(\text{Rel}(y_j, z_k) \to B_j(y_j) \vee \neg C_k(z_k))))$$

$\Rightarrow$

$$V_{(i,j),s}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j)))\big|_{z'=s}] = 1 \wedge$$
$$\forall(st \in S)(L(s, st) \to V_{(i,j),st}[\forall x_i, y_j(\text{Rel}(x_i, y_j) \to (\neg(A_i(x_i) \wedge B_j(y_j)))\big|_{z'=st}] = 1) \wedge$$
$$V_{(i,j),s'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \to B_j(y_j) \vee \neg C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall y_j, z_k(\text{Rel}(y_j, z_k) \to B_j(y_j) \vee \neg C_k(z_k)))\big|_{z'=st'}] = 1)$$

$\Rightarrow$

(we use the fact that: $(\neg a \vee \neg b) \wedge (b \vee \neg c) \Rightarrow (\neg a \vee \neg c))$

$$V_{(i,j),s'}[\forall x_i, z_j(\text{Rel}(x_i, z_j) \to (\neg A_i(x_i) \vee \neg C_k(z_k)))\big|_{z'=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s', st') \to V_{(i,j),st'}[\forall x_i, z_j(\text{Rel}(x_i, z_j) \to (\neg A_i(x_i) \vee \neg C_k(z_k)))\big|_{z'=st'}] = 1)$$

=R"

= DISJOINT(A$_i$, C$_k$)

Because we learn from §A.2.1 If L⊨ Disjoint (A$_i$, C$_k$), then L does not entail other relations.that "A.2.1 If L⊨ Disjoint (A$_i$, C$_k$), then L does not entail other relations." we can conclude that

If R= DISJOINT(A$_i$, B$_j$) and R'= IS2(B$_j$, C$_k$)

then R ∧ R' $\Rightarrow$ DISJOINT(A$_i$, C$_k$)

and R ∧ R' does not imply any other relation.

A.1.9 The case of R= Is(A$_i$, B$_j$) and R'= DISJOINT(B$_j$, C$_k$)

We know that R= Is(A$_i$, B$_j$) and R'= DISJOINT(B$_j$, C$_k$)

Then we have that

R ∧ R' = Is(A$_i$, B$_j$) ∧ DISJOINT(B$_j$, C$_k$) = Is2(B$_j$, A$_i$) ∧ DISJOINT(C$_k$, B$_j$) =

DISJOINT(C_k, B_j) $\wedge$ Is2(B_j, A_i) =(*)= DISJOINT(C_k, A_i) = DISJOINT(A_i, C_k)

We re-use the result from §A.1.8 The case when R= DISJOINT(A_i, B_j) and R'= IS2(B_j, C_k)in this calculation (in the place marked(*)).

Because we learn from §A.2.1 If L⊨ Disjoint (A_i, C_k), then L does not entail other relations.that "A.2.1 If L⊨ Disjoint (A_i, C_k), then L does not entail other relations." we can conclude that
If R= IS(A_i, B_j) and R'= DISJOINT (B_j, C_k)
then R $\wedge$ R' $\Rightarrow$ DISJOINT(A_i, C_k)
and R $\wedge$ R' does not imply any other relation.

A.1.10 The case of R= Is2(A_i, B_j) and R'= COMPATIBLE(B_j, C_k)

By looking at the definition, we see that COMPATIBLE(A_i, C_k) = COMPATIBLE(C_k, A_i)

Assuming that R= Is2(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) we have
R $\wedge$R'= Is2(A_i, B_j) $\wedge$ COMPATIBLE(B_j, C_k) = Is(B_j, A_i) COMPATIBLE(C_k, B_j) =
COMPATIBLE(C_k, B_j) $\wedge$ Is(B_j, A_i) $\Rightarrow$ COMPATIBLE(C_k, A_i) = COMPATIBLE(A_i, C_k)

In §A.2.2 If L⊨ COMPATIBLE (A_i, C_k), then L does not entail other relations.we show that if R $\wedge$R' $\Rightarrow$COMPATIBLE (A_i, C_k) and only one relation holds between A_i and C_k then R $\wedge$R' does not imply any other relation.

A.1.11 The two cases when R= DISJOINT(A_i, B_j) and R'= Is (B_j, C_k) | DISJOINT (B_j, C_k)

We now face the challenge to show that the three cases when R= DISJOINT(A_i, B_j) and R'= Is (B_j, C_k) | DISJOINT (B_j, C_k) lead to R $\wedge$R' $\Rightarrow$ ε, i.e. it does not imply any of the other relations.

A.1.11.1 Case 1. R''≠COMPATIBLE (A_i, C_k)

We show this by proof of contradiction.

It is given that R= DISJOINT(A_i, B_j) and R'= Is (B_j, C_k) | DISJOINT (B_j, C_k), and
R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k)
(we investigate later R''= COMPATIBLE (A_i, C_k))

We assume then that for every $r' \in R'$, it is the case that $R \wedge r' \Rightarrow r''$ where $r'' \in R''$.
This will expressed as:
$R \wedge R' \Rightarrow R''$

This is not true because of this counter-model (for any choice of $r' \in R'$ and $r'' \in R''$):

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=false	$I_{2,1}(B_2(b_2))$=false	$I_{3,1}(C_3(c_3))$=true
$L_{1,2}=\{A_1,a_1\}$	$L_{2,2}=\{B_2,b_2\}$	$L_{3,2}=\{C_3,c_3\}$
$D_{1,2}=\{a_1\}$	$D_{2,2}=\{b_2\}$	$D_{3,2}=\{c_3\}$
$I_{1,2}(A_1(a_1))$=true	$I_{2,2}(B_2(b_2))$=false	$I_{3,2}(C_3(c_3))$=false
$L_{1,3}=\{A_1,a_1\}$	$L_{2,3}=\{B_2,b_2\}$	$L_{3,3}=\{C_3,c_3\}$
$D_{1,3}=\{a_1\}$	$D_{2,3}=\{b_2\}$	$D_{3,3}=\{c_3\}$
$I_{1,3}(A_1(a_1))$=true	$I_{2,3}(B_2(b_2))$=false	$I_{3,3}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),2}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),3}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),3}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$

A.1.11.2 Case 2. R''=COMPATIBLE (A_i, C_k)

Now it just remains to investigate the case
R= DISJOINT(A_i, B_j) and
R'= Is (B_j, C_k) | DISJOINT (B_j, C_k), and
R''= COMPATIBLE (A_i, C_k)

We assume then that for every $r' \in R'$, it is the case that $R \wedge r' \Rightarrow R''$
This will expressed as:
$R \wedge R' \Rightarrow R''$

This is not true because of this counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=false	$I_{2,1}(B_2(b_2))$=false	$I_{3,1}(C_3(c_3))$=false
$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$

So our initial assumption was false.

We have therefore proven that
If R= DISJOINT(A_i, B_j) and
R'= Is (B_j, C_k) | DISJOINT (B_j, C_k) then
$R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

A.1.12 The two cases when R= Is2(A_i, B_j) | COMPATIBLE(A_i, B_j) and R'= DISJOINT(B_j, C_k)

We now face the challenge to show that the three cases when R= Is2(A_i, B_j) | COMPATIBLE(A_i, B_j) and R'= DISJOINT(B_j, C_k) lead to $R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

$R \wedge R'$ = (Is2(A_i, B_j) | COMPATIBLE(A_i, B_j)) $\wedge$ DISJOINT(B_j, C_k) =
Is2(A_i, B_j) $\wedge$ DISJOINT(B_j, C_k) | COMPATIBLE(A_i, B_j) $\wedge$ DISJOINT(B_j, C_k) =
Is(B_j, A_i) $\wedge$ DISJOINT(C_k, B_j) | COMPATIBLE(B_j, A_i) $\wedge$ DISJOINT(C_k, B_j) =
DISJOINT(C_k, B_j) $\wedge$ Is(B_j, A_i) | DISJOINT(C_k, B_j) |COMPATIBLE(B_j, A_i) $\Rightarrow \varepsilon$

The reason for this is that these three cases have already been investigated in §A.1.11 The two cases when R= DISJOINT(A_i, B_j) and
R'= Is (B_j, C_k) | DISJOINT (B_j, C_k). Also, in A.1.17 we motivate the (obvious) fact that if R(A,B) $\Rightarrow \varepsilon$ then R(B,A) $\Rightarrow \varepsilon$ if R is one of the five relations.

A.1.13. The case of R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k)

We want to prove that
If R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) then

$R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

We show this by proof of contradiction.

It is given that R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k)

and R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE(A_i, C_k)

A.1.13.1 Case 1, $R'' \neq$ DISJOINT (A_i, C_k)

R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k)

and R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | COMPATIBLE(A_i, C_k)

We assume then that for every $r' \in R'$, it is the case that $R \wedge r' \Rightarrow r''$ where $r'' \in R''$.

This will expressed as:

$R \wedge R' \Rightarrow R''$

DISJOINT(A_i, B_j) $\wedge$ COMPATIBLE(B_j, C_k) $\Rightarrow$ R''

This is not true because of this counter-model (for any choice of $r'' \in R''$):

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=false	$I_{3,1}(C_3(c_3))$=false
$L_{1,2}=\{A_1,a_1\}$	$L_{2,2}=\{B_2,b_2\}$	$L_{3,2}=\{C_3,c_3\}$
$D_{1,2}=\{a_1\}$	$D_{2,2}=\{b_2\}$	$D_{3,2}=\{c_3\}$
$I_{1,2}(A_1(a_1))$=false	$I_{2,2}(B_2(b_2))$=true	$I_{3,2}(C_3(c_3))$=true
$L_{1,3}=\{A_1,a_1\}$	$L_{2,3}=\{B_2,b_2\}$	$L_{3,3}=\{C_3,c_3\}$
$D_{1,3}=\{a_1\}$	$D_{2,3}=\{b_2\}$	$D_{3,3}=\{c_3\}$
$I_{1,3}(A_1(a_1))$=false	$I_{2,3}(B_2(b_2))$=true	$I_{3,3}(C_3(c_3))$=false

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),2}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),3}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),3}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$

A.1.13.2 Case 2, $R''=$ DISJOINT (A_i, C_k)

R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) and R''= DISJOINT(B_j, C_k)

We assume that $R \wedge R' \Rightarrow R''$

This is not true because of this counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=false	$I_{3,1}(C_3(c_3))$=true
$L_{1,2}=\{A_1,a_1\}$	$L_{2,2}=\{B_2,b_2\}$	$L_{3,2}=\{C_3,c_3\}$
$D_{1,2}=\{a_1\}$	$D_{2,2}=\{b_2\}$	$D_{3,2}=\{c_3\}$
$I_{1,2}(A_1(a_1))$=false	$I_{2,2}(B_2(b_2))$=true	$I_{3,2}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),2}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$

We have therefore proved that

If R= DISJOINT(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) then

$R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

A.1.14 The case of R= IS(A_i, B_j) and R'= IS2(B_j, C_k)

We now need to show that the three cases when

R= IS(A_i, B_j) and R'= IS2(B_j, C_k) lead to $R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

We show this by a proof of contradiction.

Assume that $R \wedge R' \Rightarrow R''$

and R"= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE(B_j, C_k)

This is investigated by splitting this into two cases.

A.1.14.1 Case 1, R´´≠ DISJOINT (A_i, C_k)

R= IS(A_i, B_j) and R'= IS2(B_j, C_k)

and R"= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | COMPATIBLE(A_i, C_k)

Assume that $R \wedge R' \Rightarrow R''$

This is not true because of this counter-model (for any choice of $r'' \in R''$):

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=true	$I_{3,1}(C_3(c_3))$=false

$L_{1,2}=\{A_1,a_1\}$	$L_{2,2}=\{B_2,b_2\}$	$L_{3,2}=\{C_3,c_3\}$
$D_{1,2}=\{a_1\}$	$D_{2,2}=\{b_2\}$	$D_{3,2}=\{c_3\}$
$I_{1,2}(A_1(a_1))$=false	$I_{2,2}(B_2(b_2))$=false	$I_{3,2}(C_3(c_3))$=false
$L_{1,3}=\{A_1,a_1\}$	$L_{2,3}=\{B_2,b_2\}$	$L_{3,3}=\{C_3,c_3\}$
$D_{1,3}=\{a_1\}$	$D_{2,3}=\{b_2\}$	$D_{3,3}=\{c_3\}$
$I_{1,3}(A_1(a_1))$=false	$I_{2,3}(B_2(b_2))$=true	$I_{3,3}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),2}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),2}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),3}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),3}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$

A.1.14.2 Case 2, R´´= DISJOINT (A_i, C_k)

R= IS(A_i, B_j) and R'= IS2(B_j, C_k) and R"= DISJOINT(A_i, C_k)

Assume that $R \wedge R' \Rightarrow R''$

This is not true because of this counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=true	$I_{3,1}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$

We have therefore proven that if

R= IS(A_i, B_j) and R'= IS2(B_j, C_k) then

$R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

A.1.15 The case of R= IS2(A_i, B_j) and R'= IS(B_j, C_k)

We now need to show that the three cases when

R= IS2(A_i, B_j) and R'= IS(B_j, C_k) lead to $R \wedge R' \Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

We show this by a proof of contradiction.

Assume that $R \wedge R' \Rightarrow R''$

and R"= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE(A_i, C_k)

This is investigated by splitting this into two cases.

A.1.15.1 Case 1, R´´≠ DISJOINT (A_i, C_k)

R= IS2(A_i, B_j) and R'= IS(B_j, C_k)

and R"= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | COMPATIBLE(A_i, C_k)

Assume that R $\wedge$ R' $\Rightarrow$ R"

This is not true because of this counter-model (for any choice of r"$\in$R"):

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=false	$I_{2,1}(B_2(b_2))$=false	$I_{3,1}(C_3(c_3))$=false
$L_{1,2}=\{A_1,a_1\}$	$L_{2,2}=\{B_2,b_2\}$	$L_{3,2}=\{C_3,c_3\}$
$D_{1,2}=\{a_1\}$	$D_{2,2}=\{b_2\}$	$D_{3,2}=\{c_3\}$
$I_{1,2}(A_1(a_1))$=true	$I_{2,2}(B_2(b_2))$=false	$I_{3,2}(C_3(c_3))$=false
$L_{1,3}=\{A_1,a_1\}$	$L_{2,3}=\{B_2,b_2\}$	$L_{3,3}=\{C_3,c_3\}$
$D_{1,3}=\{a_1\}$	$D_{2,3}=\{b_2\}$	$D_{3,3}=\{c_3\}$
$I_{1,3}(A_1(a_1))$=false	$I_{2,3}(B_2(b_2))$=false	$I_{3,3}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),2}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),2}(Rel)=\{(a_1, c_3)\}$
$I_{(1,2),3}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),3}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),3}(Rel)=\{(a_1, c_3)\}$

A.1.15.2 Case 2, R´´= DISJOINT (A_i, C_k)

R= IS2(A_i, B_j) and R'= IS(B_j, C_k) and R"= DISJOINT(A_i, C_k)

Assume that R $\wedge$ R' $\Rightarrow$ R"

This is not true because of this counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=true	$I_{3,1}(C_3(c_3))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$

We have therefore proven that if

R= Is2(A_i, B_j) and R'= Is(B_j, C_k) then

R $\wedge$R' $\Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

A.1.16 The case of R= Is(A_i, B_j) and R'= Compatible(B_j, C_k)

We now need to show that the three cases when

R= Is(A_i, B_j) and R'= Compatible(B_j, C_k) lead to R $\wedge$R' $\Rightarrow \varepsilon$, i.e. it does not imply any of the other relations.

We show this by a proof of contradiction.

Assume that R $\wedge$ R' $\Rightarrow$ R"

and R"= Cor(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | Disjoint (A_i, C_k) | Compatible(A_i, C_k)

This is investigated by splitting this into two cases.

A.1.16.1 Case 1, R´´$\neq$ Disjoint (A_i, C_k)

R= Is(A_i, B_j) and R'= Compatible(B_j, C_k)

and R"= Cor(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | Compatible(A_i, C_k)

Assume that R $\wedge$ R' $\Rightarrow$ R"

This is not true because of this counter-model (for any choice of r"$\in$R"):

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}	$L_{3,1}$={C_3,c_3}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}	$D_{3,1}$={c_3}
$I_{1,1}$(A_1(a_1))=false	$I_{2,1}$(B_2(b_2))=true	$I_{3,1}$(C_3(c_3))=true
$L_{1,2}$={A_1,a_1}	$L_{2,2}$={B_2,b_2}	$L_{3,2}$={C_3,c_3}
$D_{1,2}$={a_1}	$D_{2,2}$={b_2}	$D_{3,2}$={c_3}
$I_{1,2}$(A_1(a_1))=false	$I_{2,2}$(B_2(b_2))=false	$I_{3,2}$(C_3(c_3))=true
$L_{1,3}$={A_1,a_1}	$L_{2,3}$={B_2,b_2}	$L_{3,3}$={C_3,c_3}
$D_{1,3}$={a_1}	$D_{2,3}$={b_2}	$D_{3,3}$={c_3}
$I_{1,3}$(A_1(a_1))=true	$I_{2,3}$(B_2(b_2))=true	$I_{3,3}$(C_3(c_3))=false

$I_{(1,2),1}$(Rel)={(a_1, b_2)}	$I_{(2,3),1}$(Rel)={(b_2, c_3)}	$I_{(1,3),1}$(Rel)={(a_1, c_3)}
$I_{(1,2),2}$(Rel)={(a_1, b_2)}	$I_{(2,3),2}$(Rel)={(b_2, c_3)}	$I_{(1,3),2}$(Rel)={(a_1, c_3)}
$I_{(1,2),3}$(Rel)={(a_1, b_2)}	$I_{(2,3),3}$(Rel)={(b_2, c_3)}	$I_{(1,3),3}$(Rel)={(a_1, c_3)}

A.1.16.2 Case 2, R´´= DISJOINT (A_i, C_k)

R= IS(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) and R''= DISJOINT(A_i, C_k)

Assume that R $\wedge$ R' $\Rightarrow$ R''

This is not true because of this counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	$L_{3,1}=\{C_3,c_3\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	$D_{3,1}=\{c_3\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=true	$I_{3,1}(C_3(c_3))$=true
$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$	$I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$	$I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$

We have therefore proven that if

R= IS(A_i, B_j) and R'= COMPATIBLE(B_j, C_k) then

R $\wedge$R' $\Rightarrow$ ε, i.e. it does not imply any of the other relations.

A.1.17 The case of R= COMPATIBLE (A_i, B_j) and R'= IS2(B_j, C_k)

We now need to show that when

R= COMPATIBLE (A_i, B_j) and R'= IS2(B_j, C_k) then R $\wedge$R' $\Rightarrow$ ε, i.e. it does not imply any of the other relations.

We show this by a proof of contradiction.

Assume that R $\wedge$ R' $\Rightarrow$ R''

and R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE(A_i, C_k)

R $\wedge$ R' = COMPATIBLE (A_i, B_j) $\wedge$ IS2(B_j, C_k) = COMPATIBLE (B_j, A_i) $\wedge$ IS(C_k, B_j)=

IS(C_k, B_j) $\wedge$ COMPATIBLE (B_j, A_i) $\nrightarrow$

COR(C_k, A_i) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (C_k , A_i) | COMPATIBLE(C_k ,A_j)

$\Rightarrow$

R'' $\nrightarrow$ COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE(A_j, C_k)

$\Rightarrow$

R''$\rightarrow\varepsilon$

We have utilised the symmetry of several relations.

A.1.18 The case of R= COMPATIBLE(A_i, B_j) and R'= COMPATIBLE (B_j, C_k)

R= COMPATIBLE (A_i, B_j) and R'= COMPATIBLE(B_j, C_k)
and R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | COMPATIBLE(B_j, C_k) | DISJOINT(B_j, C_k)
Assume that R $\wedge$ R' $\Rightarrow$ R''

We divide this investigation in two parts, and firstly assume that R´´≠ DISJOINT (A_i, C_k).

A.1.18.1 Case 1, R´´≠ DISJOINT (A_i, C_k)

Given that R= COMPATIBLE (A_i, B_j) and R'= COMPATIBLE(B_j, C_k)
and R''= COR(A_i, C_k) | Is(A_i, C_k) | Is2(A_i, C_k) | COMPATIBLE(A_i, C_k)

Assume that R $\wedge$ R' $\Rightarrow$ R''

This is not true because of this counter-model (for any choice of r''∈R''):

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}	$L_{3,1}$={C_3,c_3}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}	$D_{3,1}$={c_3}
$I_{1,1}$(A_1(a_1))=false	$I_{2,1}$(B_2(b_2))=true	$I_{3,1}$(C_3(c_3))=true
$L_{1,2}$={A_1,a_1}	$L_{2,2}$={B_2,b_2}	$L_{3,2}$={C_3,c_3}
$D_{1,2}$={a_1}	$D_{2,2}$={b_2}	$D_{3,2}$={c_3}
$I_{1,2}$(A_1(a_1))=true	$I_{2,2}$(B_2(b_2))=true	$I_{3,2}$(C_3(c_3))=false

$I_{(1,2),1}$(Rel)={(a_1, b_2)}	$I_{(2,3),1}$(Rel)={(b_2, c_3)}	$I_{(1,3),1}$(Rel)={(a_1, c_3)}
$I_{(1,2),2}$(Rel)={(a_1, b_2)}	$I_{(2,3),2}$(Rel)={(b_2, c_3)}	$I_{(1,3),2}$(Rel)={(a_1, c_3)}

A.1.18.2 Case 2, R´´= DISJOINT (A_i, C_k)

R= COMPATIBLE (A_i, B_j) and R'= COMPATIBLE(B_j, C_k) and R''= DISJOINT(A_i, C_k)
Assume that R $\wedge$ R' $\Rightarrow$ R''

This is not true because of this counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}	$L_{3,1}$={C_3,c_3}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}	$D_{3,1}$={c_3}
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=true	$I_{3,1}$(C_3(c_3))=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$ $I_{(2,3),1}(Rel)=\{(b_2, c_3)\}$ $I_{(1,3),1}(Rel)=\{(a_1, c_3)\}$

We have therefore proven that if
R= COMPATIBLE (A_i, B_j) and R'= COMPATIBLE(B_j, C_k) then
R $\wedge$R' $\Rightarrow$ ε, i.e. it does not imply any of the other relations.

A.1.19 Conclusion of this section

We have now investigated all 25 cases in Table 3 and show that our mechanical procedure is correct and complete with regard to the semantics, as regards calculating the cases in this table.

The following section (§A.2 Proofs of the form: "If L$\models$ Rel (A_i, C_k), then L does not entail other relations" contains certain proofs that are not interesting as such, but that have been utilized in proving theorem 1.

A.2 Proofs of the form: "If L$\models$ Rel (A_i, C_k), then L does not entail other relations"

We will now prove the following:

- If L$\models$ Disjoint(A_i, C_k), then L does not entail other relations.
- If L$\models$ COMPATIBLE(A_i, C_k), then L does not entail other relations.
- If L$\models$ Is(A_i, C_k) and L$\not\models$Cor(A_i, C_k) then L does not entail other relations.
- If L$\models$ Is2(A_i, C_k) and L$\not\models$Cor(A_i, C_k) then L does not entail other relations.
- If L$\models$ Cor(A_i, C_k) then L$\models$ Is(A_i, C_k) and L$\models$ Is2(A_i, C_k) but L does not entail other relations.

A.2.1 If L$\models$ Disjoint (A_i, C_k), then L does not entail other relations.

We need to prove that if
R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\models$ DISJOINT (A_i, C_k) and
R= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k) and

R'= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

Then $R(A_i, B_j) \wedge R'(B_j, C_k) \nvDash$ Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | COMPATIBLE (B_j, C_k)

A.2.1.1 The case of L⊨ Cor (A_i, C_k)

L= $R(A_i, B_j) \wedge R'(B_j, C_k)$ and L is consistent, i.e. not logically false.

We known that L⊨ DISJOINT (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.

Assume now that also L⊨ Cor (A_i, C_k) holds.

We then have:

$$L \rightarrow \mathrm{DISJOINT}(\mathrm{A_i},\mathrm{C_k}) \wedge L \rightarrow \mathrm{COR}(\mathrm{A_i},\mathrm{C_k})$$
$$\Rightarrow$$
$$(\neg L \vee \mathrm{DISJOINT}(\mathrm{A_i},\mathrm{C_k})) \wedge (\neg L \vee \mathrm{COR}(\mathrm{A_i},\mathrm{C_k}))$$
$$\Rightarrow$$
$$(\neg L) \vee (\neg L \wedge \mathrm{COR}(\mathrm{A_i},\mathrm{C_k})) \vee (\neg L \wedge \mathrm{DISJOINT}(\mathrm{A_i},\mathrm{C_k})) \vee$$
$$(\mathrm{DISJOINT}(\mathrm{A_i},\mathrm{C_k}) \wedge \mathrm{COR}(\mathrm{A_i},\mathrm{C_k}))$$
$$\Rightarrow$$
$$(\mathrm{DISJOINT}(\mathrm{A_i},\mathrm{C_k}) \wedge \mathrm{COR}(\mathrm{A_i},\mathrm{C_k}))$$

We now see that two relations hold between A_i and C_k, whereas we assumed that only one relationship hold between A_i and C_k. The question is then if

Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) $\rightarrow$ DISJOINT (A_i, C_k) $\wedge$ Cor (A_i, C_k)

Because Cor (A_i, C_k) $\nvDash$ DISJOINT (A_i, C_k) (see §A.3.1), Is (A_i, C_k) $\nvDash$ DISJOINT (A_i, C_k) (see §A.3.2), Is2 (A_i, C_k) $\nvDash$ DISJOINT (A_i, C_k) (see §A.3.20), COMPATIBLE (A_i, C_k) $\nvDash$ DISJOINT (A_i, C_k) (see §A.3.6) we conclude that the equation above is not true. Therefore if L⊨ DISJOINT (A_i, C_k) it cannot be the case that L ⊨ Cor (A_i, C_k).

A.2.1.2 The case of L ⊨ Is (A_i, C_k)

L= $R(A_i, B_j) \wedge R'(B_j, C_k)$ and L is consistent, i.e. not logically false.

We known that L⊨DISJOINT (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.

Assume now that also L⊨ Is (A_i, C_k) holds.

This leads to:

$$(\mathrm{DISJOINT}(A_i, C_k) \wedge \mathrm{IS}(A_i, C_k))$$

relations hold between A_i and C_k, whereas we assumed that only one relationship hold between A_i and C_k. The question is then if

Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) ⊨
DISJOINT (A_i, C_k) ∧ Is (A_i, C_k)

Because Cor (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.1), Is (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.2), Is2 (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.20), DISJOINT (A_i, C_k) ⊭ Is (A_i, C_k) (see §A.3.12), COMPATIBLE (A_i, C_k) ⊭DISJOINT (A_i, C_k) (see §A.3.6) we conclude that the equation above is not true. Therefore if L⊨ DISJOINT (A_i, C_k) it cannot be the case that L ⊨ Is (A_i, C_k).

A.2.1.3 The case of L ⊨ Is2 (A_i, C_k)

If R"= Is2 (B_j, C_k) then the same method of reasoning will lead to:

$$(\mathrm{DISJOINT}(A_i, C_k) \wedge \mathrm{IS2}(A_i, C_k))$$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship hold between A_i and C_k. The question is then if

Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) ⊨
DISJOINT (A_i, C_k) ∧ Is2 (A_i, C_k)

Because Cor (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.1), Is (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.2), Is2 (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.20), DISJOINT (A_i, C_k) ⊭ Is2 (A_i, C_k) (see §A.3.15), COMPATIBLE (A_i, C_k) ⊭ DISJOINT (A_i, C_k) (see §A.3.6) we conclude that the equation above is not true. Therefore if L⊨ DISJOINT (A_i, C_k) it cannot be the case that L ⊨ Is2 (A_i, C_k).

I.e. the conclusion is not true in general.

A.2.1.3 The case of L $\models$ COMPATIBLE(A_i, C_k)

If R''= Compatible (B_j, C_k) then the same method of reasoning will lead to:

$$(\text{DISJOINT}(A_i, C_k) \wedge \text{COMPATIBLE}(A_i, C_k))$$

This time, it is easy to show that it is impossible, because §A.3.6 and §A.3.19 prove that these two relations mutually contradict each other.

To conclude this section we have proved the following.
If R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\models$ DISJOINT (A_i, C_k) and
R= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k) and
R'= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)
Then R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\not\models$Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | COMPATIBLE (B_j, C_k)

A.2.2 If L$\models$ COMPATIBLE (A_i, C_k), then L does not entail other relations.

We need to prove that if
R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\models$ COMPATIBLE (A_i, C_k) and
R= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k) and
R'= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)
Then R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\not\models$ Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k)

A.2.2.1 The case of L $\models$ IS (A_i, C_k)

L= R(A_i, B_j) $\wedge$ R'(B_j, C_k) and L is consistent.
We know that L$\models$ COMPATIBLE (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.
Assume now that also L$\models$ Is (A_i, C_k) holds.

This will lead to COMPATIBLE (A_i, C_k) $\wedge$ Is (A_i, C_k).
We now see that two relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. The question is then if
Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) $\models$ COMPATIBLE (A_i, C_k) $\wedge$ Is (A_i, C_k).
Because Cor (A_i, C_k) $\not\models$ COMPATIBLE (A_i, C_k) (see §A.3.16), Is (A_i, C_k) $\not\models$ COMPATIBLE (A_i, C_k) (see §A.3.17), Is2 (A_i, C_k) $\not\models$ COMPATIBLE(A_i, C_k) (see §A.3.18), DISJOINT (A_i, C_k) $\not\models$ Is

(A_i, C_k) (see §A.3.12), COMPATIBLE (A_i, C_k) $\nvDash$ IS (A_i, C_k) (see §A.3.11) we conclude that the equation above is not true.
Therefore if L$\vDash$ COMPATIBLE (A_i, C_k) it cannot be the case that L $\vDash$Is (A_i, C_k).

A.2.2.2 The case of L $\vDash$ IS2 (A_i, C_k)

If R"= Is2 (B_j, C_k) then the same method of reasoning will lead to:

$$(\mathrm{COMPATIBLE}\left(\mathrm{A_i},\mathrm{C_k}\right)\wedge \mathrm{IS2}\left(\mathrm{A_i},\mathrm{C_k}\right))$$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship hold between A_i and C_k. The question is then if
Cor (A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) $\vDash$
COMPATIBLE (A_i, C_k) $\wedge$ Is2 (A_i, C_k)

Because Cor(A_i, C_k) $\nvDash$ COMPATIBLE (A_i, C_k) (see §A.3.16), Is(A_i, C_k) $\nvDash$ COMPATIBLE (A_i, C_k) (see §A.3.17), Is2 (A_i, C_k) $\nvDash$ COMPATIBLE(A_i, C_k) (see §A.3.18), DISJOINT (A_i, C_k) $\nvDash$ Is2(A_i, C_k) (see §A.3.15), COMPATIBLE (A_i, C_k) $\nvDash$ IS2(A_i, C_k) (see §A.3.14) we conclude that the equation above is not true.
Therefore if L$\vDash$ COMPATIBLE (A_i, C_k) it cannot be the case that L $\vDash$ Is2(A_i, C_k).

A.2.2.3 The case of L $\vDash$ COR (A_i, C_k)

In §0 we showed that it is not possible that

$$L\rightarrow \mathrm{COMPATIBLE}\left(\mathrm{A_i},\mathrm{C_k}\right)\wedge L\rightarrow \mathrm{COR}\left(\mathrm{A_i},\mathrm{C_k}\right)$$

assuming that only one relation holds between A_i and C_k, and due to reasons of symmetry we reuse that conclusion.
Therefore, if L$\vDash$ COMPATIBLE (A_i, C_k) it cannot be the case that L $\vDash$ Cor(A_i, C_k).

A.2.2.4 The case of L $\vDash$ DISJOINT (A_i, C_k)

If
L$\vDash$ COMPATIBLE (A_i, C_k) and L$\vDash$ DISJOINT (A_i, C_k) then

$$(\mathrm{COMPATIBLE}\left(\mathrm{A_i},\mathrm{C_k}\right)\wedge \mathrm{DISJOINT}\left(\mathrm{A_i},\mathrm{C_k}\right))$$

However, this is not possible because according to (§0 and §0) these relations always contradict each other.
Therefore, if L$\vDash$ COMPATIBLE (A_i, C_k) it cannot be the case that L$\vDash$ DISJOINT (A_i, C_k).

To conclude this section we have proved the following.

If $R(A_i, B_j) \wedge R'(B_j, C_k) \vDash$ COMPATIBLE (A_i, C_k) and

R= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k) and

R'= Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

Then $R(A_i, B_j) \wedge R'(B_j, C_k) \nrightarrow$ Cor (B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k)

A.2.3 If L⊨ IS (A_i, C_k) and L⊭COR(A_i, C_k) then L does not entail other relations.

We need to prove that if

$R(A_i, B_j) \wedge R'(B_j, C_k) \vDash$ IS(A_i, C_k) and

R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and

R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

and ¬(R= COR$(A_i, B_j) \wedge$ R'= COR(B_j, C_k))

then $R(A_i, B_j) \wedge R'(B_j, C_k) \nvDash$ Cor (B_j, C_k) | COMPATIBLE (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k)

A.2.3.1 The case of L ⊨ COMPATIBLE (A_i, C_k)

L= $R(A_i, B_j) \wedge R'(B_j, C_k)$ and L is consistent, i.e. not logically false.

We know that L⊨ COMPATIBLE (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.

Assume now that also L⊨ Is (A_i, C_k) holds.

Using the earlier proof pattern this will lead to IS $(A_i, C_k) \wedge$ COMPATIBLE (A_i, C_k).

We now see that two relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. In §A.2.2.1 we have already shown that this is not possible (if we utilize the symmetry of ∧).

Therefore if L⊨ IS (A_i, C_k) and L⊭ Cor(A_i, C_k) it cannot be the case that L ⊨ COMPATIBLE (A_i, C_k).

A.2.3.2 The case of L ⊨ IS2 (A_i, C_k)

If R"= Is2 (A_i, C_k) then the same method of reasoning will lead to:

$$(\mathrm{IS}(A_i, C_k) \wedge \mathrm{IS2}(A_i, C_k))$$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. The question is then if

Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE $(A_i, C_k) \vDash$
IS $(A_i, C_k) \wedge$ IS2 (A_i, C_k)

Because IS$(A_i, C_k) \nvDash$ IS2(A_i, C_k) (see §A.3.10), Is2$(A_i, C_k) \nvDash$ IS(A_i, C_k) (see §A.3.9), DISJOINT$(A_i, C_k) \nvDash$ IS(A_i, C_k) (see §A.3.12), COMPATIBLE $(A_i, C_k) \nvDash$ Is(A_i, C_k) (see §A.3.11), we conclude that the equation above is not true.
Therefore if L$\vDash$ IS (A_i, C_k) and L$\nvDash$ Cor(A_i, C_k) it cannot be the case that L $\vDash$ Is2(A_i, C_k).

A.2.3.3 The case of L $\vDash$ COR (A_i, C_k)

The reason why we excluded this case, is that it is possible for
L$\vDash$ IS(A_i, C_k) and L$\vDash$ COR(A_i, C_k) to hold at the same time.

A.2.3.4 The case of L $\vDash$ DISJOINT (A_i, C_k)

If R''= DISJOINT(A_i, C_k) then the same method of reasoning as described before will lead to:

$$(\mathrm{IS}(A_i, C_k) \wedge \mathrm{DISJOINT}(A_i, C_k))$$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k.
Due to reasons of symmetry of $\wedge$ we reuse the result from §0 that shows that this is not possible.

Therefore if L$\vDash$ IS (A_i, C_k) and L$\nvDash$ Cor(A_i, C_k) it cannot be the case that L $\vDash$ DISJOINT(A_i, C_k).

To conclude this section, we have proved the following.
R$(A_i, B_j) \wedge$ R'$(B_j, C_k) \vDash$ IS(A_i, C_k) and
R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and
R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)
and $\neg$(R= COR$(A_i, B_j) \wedge$ R'= COR(B_j, C_k))
then R$(A_i, B_j) \wedge$ R'$(B_j, C_k) \nvDash$ Cor (A_i, C_k) | COMPATIBLE (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k)

A.2.4 If L$\vDash$ IS2 (A_i, C_k) and and L$\nvDash$COR(A_i, C_k) then L does not entail other relations.

We need to prove that if

$R(A_i, B_j) \wedge R'(B_j, C_k) \vDash$ IS2(A_i, C_k) and

R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and

R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

and $\neg$(R= COR$(A_i, B_j) \wedge$ R'= COR(B_j, C_k))

then $R(A_i, B_j) \wedge R'(B_j, C_k) \nvDash$ Cor (B_j, C_k) | COMPATIBLE (A_i, C_k) | Is (A_i, C_k) | DISJOINT (A_i, C_k)

A.2.4.1 The case of L $\vDash$ COMPATIBLE (A_i, C_k)

L= $R(A_i, B_j) \wedge R'(B_j, C_k)$ and L is consistent, i.e. not logically false.
We know that L$\vDash$ COMPATIBLE (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.
Assume now that also L$\vDash$ Is2(A_i, C_k) holds.

This will lead to IS2$(A_i, C_k) \wedge$ COMPATIBLE (A_i, C_k).
We now see that two relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. In §A.2.2.1 we have already shown that this is not possible (if we utilize the symmetry of $\wedge$).

Therefore if L$\vDash$ IS2(A_i, C_k) and L$\nvDash$ Cor(A_i, C_k) it cannot be the case that L $\vDash$ COMPATIBLE (A_i, C_k).

A.2.4.2 The case of L $\vDash$ IS (A_i, C_k)

If R"= IS(A_i, C_k) then the same method of reasoning as described before will lead to:

$$(\mathrm{IS2}(A_i, C_k) \wedge \mathrm{IS}(A_i, C_k))$$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k.
Due to reasons of symmetry of $\wedge$ we reuse the result from §A.2.3.2 that shows that this is not possible.

A.2.4.3 The case of L $\vDash$ COR (A_i, C_k)

The reason why we excluded this case, is that it is possible for
L$\vDash$ IS2(A_i, C_k) and L$\vDash$ COR(A_i, C_k) to hold at the same time.

A.2.4.4 The case of L $\vDash$ DISJOINT (A_i, C_k)

If R"= IS(A_i, C_k) then the same method of reasoning as described before will lead to:

$(\mathrm{IS2}(A_i, C_k) \wedge \mathrm{DISJOINT}(A_i, C_k))$

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k.

Due to reasons of symmetry of $\wedge$ we reuse the result from §A.2.1.3. that shows that this is not possible.

To conclude this section, we have proved that if

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\vDash$ IS2(A_i, C_k) and

R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and

R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

and $\neg$(R= COR(A_i, B_j) $\wedge$ R'= COR(B_j, C_k))

then R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\nvDash$ Cor (A_i, C_k) | COMPATIBLE (A_i, C_k) | Is (A_i, C_k) | DISJOINT (A_i, C_k)

A.2.5 If L$\vDash$ Cor(A_i, C_k) then L$\vDash$ Is(A_i, C_k) and L$\vDash$ Is2(A_i, C_k) but L does not entail other relations.

We need to prove that if

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\vDash$ COR(A_i, C_k) and

R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and

R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

then R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\vDash$IS(A_i, C_k) and

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\vDash$ IS2(A_i, C_k)

But also then

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\nvDash$ COMPATIBLE (A_i, C_k) | DISJOINT (A_i, C_k)

A.2.5.1 The case of L $\vDash$ IS (A_i, C_k)

Given the assumptions above, we know that

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\vDash$ IS(A_i, C_k)

Because COR(A_i, C_k) $\vDash$ IS(A_i, C_k) according to §A.3.8.

A.2.5.2 The case of L $\models$ IS2 (A_i, C_k)

Given the assumptions in §A.2.5, we know that

R(A_i, B_j) $\wedge$ R'(B_j, C_k) $\models$ IS2(A_i, C_k)

Because COR(A_i, C_k) $\models$ IS2(A_i, C_k) according to §A.3.13.

A.2.5.3 The case of L $\models$ COMPATIBLE (A_i, C_k)

L= R(A_i, B_j) $\wedge$ R'(B_j, C_k) and L is consistent, i.e. not logically false.

We know that L$\models$ COR (A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.

Assume now that also L$\models$ COMPATIBLE (A_i, C_k) holds.

This will lead to COR(A_i, C_k) $\wedge$ COMPATIBLE (A_i, C_k).

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. The question is then if

COR(A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) $\models$

COR(A_i, C_k) $\wedge$ COMPATIBLE (A_i, C_k)

Because COR(A_i, C_k) $\not\models$COMPATIBLE(A_i, C_k) (see §A.3.16), Is(A_i, C_k) $\not\models$ COMPATIBLE (A_i, C_k) (see §A.3.17), IS2(A_i, C_k) $\not\models$ COMPATIBLE (A_i, C_k) (see §A.3.18), DISJOINT(A_i, C_k) $\not\models$ COMPATIBLE(A_i, C_k) (see §A.3.19), and COMPATIBLE(A_i, C_k) $\not\models$ COR(A_i, C_k) (see §A.3.7) we conclude that the equation above is not true.

Therefore if L$\models$COR(A_i, C_k) (and only one relation holds between A_i and C_k) it cannot be the case that L $\models$ COMPATIBLE(A_i, C_k).

A.2.5.4 The case of L $\models$ DISJOINT (A_i, C_k)

L= R(A_i, B_j) $\wedge$ R'(B_j, C_k) and L is consistent, i.e. not logically false.

We know that L$\models$ COR(A_i, C_k), and that only one relation holds between A_i and B_j, B_j and C_k, and A_i and C_k.

Assume now that also L$\models$ DISJOINT(A_i, C_k) holds.

This will lead to COR(A_i, C_k) $\wedge$ DISJOINT (A_i, C_k).

I.e. these relations hold between A_i and C_k, whereas we assumed that only one relationship holds between A_i and C_k. The question is then if

COR(A_i, C_k) | Is (A_i, C_k) | Is2 (A_i, C_k) | DISJOINT (A_i, C_k) | COMPATIBLE (A_i, C_k) ⊨

COR(A_i, C_k) ∧ DISJOINT (A_i, C_k)

Because COR(A_i, C_k) ⊭ DISJOINT(A_i, C_k) (see §A.3.1), Is(A_i, C_k) ⊭ DISJOINT(A_i, C_k) (see §A.3.2), IS2(A_i, C_k) ⊭ DISJOINT(A_i, C_k) (see §A.3.20), DISJOINT(A_i, C_k) ⊭ COR(A_i, C_k) (see §A.3.5), and COMPATIBLE(A_i, C_k) ⊭ DISJOINT(A_i, C_k) (see §A.3.6) we conclude that the equation above is not true.

Therefore if L⊨ COR(A_i, C_k) (and only one relation holds between A_i and C_k) it cannot be the case that L ⊨ DISJOINT(A_i, C_k).

To conclude this section, we have proved that if

R(A_i, B_j) ∧ R'(B_j, C_k) ⊨ COR(A_i, C_k) where

R= COR(A_i, B_j) | Is (A_i, B_j) | Is2 (A_i, B_j) | DISJOINT (A_i, B_j) | COMPATIBLE (A_i, B_j) and

R'= COR(B_j, C_k) | Is (B_j, C_k) | Is2 (B_j, C_k) | DISJOINT (B_j, C_k) | COMPATIBLE (B_j, C_k)

then R(A_i, B_j) ∧ R'(B_j, C_k) ⊨ IS(A_i, C_k) and

R(A_i, B_j) ∧ R'(B_j, C_k) ⊨ IS2(A_i, C_k)

But also then

R(A_i, B_j) ∧ R'(B_j, C_k) ⊭ COMPATIBLE (A_i, C_k) | DISJOINT (A_i, C_k)

A.3 Entailment between relations

A.3.1 Does Cor(A_i, B_j) ⊨ DISJOINT(A_i, B_j)

Does Cor(A_i, B_j) ⊨ DISJOINT(A_i, B_j) hold?

The first case is when both sides are ontology mappings.

In this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

The second case is when both are relations within the same ontology.

Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$

$D_{1,1}=\{a_1,b_1\}$

$I_{1,1}(A_1(a_1))=true$ $\qquad$ $I_{1,1}(B_1(b_1))=true$

Therefore, $Cor(A_i, B_j) \models$ DISJOINT(A_i, B_j) does not hold.

A.3.2 Does $Is(A_i, B_j) \models$ DISJOINT(A_i, B_j) hold?

Does $Is(A_i, B_j) \models$ DISJOINT(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))=true$	$I_{2,1}(B_2(b_2))=true$

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

The second case is when both are relations within the same ontology.

Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$

$D_{1,1}=\{a_1,b_1\}$

$I_{1,1}(A_1(a_1))=true$ $\qquad$ $I_{1,1}(B_1(b_1))=true$

Therefore, $Is(A_i, B_j) \models$ DISJOINT(A_i, B_j) does not hold.

A.3.3 Does Is(A_i, B_j) ⊨ COR(A_i, B_j) hold?

Does Is(A_i, B_j) ⊨ DISJOINT(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=true

$I_{(1,2),1}$(Rel)={(a_1, b_2)}

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}$={A_1,a_1, B_1,b_1}
$D_{1,1}$={a_1,b_1}

$I_{1,1}$(A_1(a_1))=true	$I_{1,1}$(B_1(b_1))=true

Therefore, Is(A_i, B_j) ⊨ DISJOINT(A_i, B_j) does not hold.

A.3.4 Does Is2(A_i, B_j) ⊨ COR(A_i, B_j) hold?

Does Is2(A_i, B_j) ⊨ COR(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}	
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}	
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=false	$I_{(1,2),1}$(Rel)={(a_1, b_2)}

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$

$D_{1,1}=\{a_1,b_1\}$

$I_{1,1}(A_1(a_1))=\text{true}$ $\qquad$ $I_{1,1}(B_1(b_1))=\text{false}$

Therefore, Is2(A_i, B_j) ⊨ DISJOINT(A_i, B_j) does not hold.

A.3.5 Does DISJOINT(A_i, B_j) ⊨Cor(A_i, B_j) hold?

Does DISJOINT (A_i, B_j) ⊨ COR(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$	
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$	
$I_{1,1}(A_1(a_1))=\text{false}$	$I_{2,1}(B_2(b_2))=\text{true}$	$I_{(1,2),1}(\text{Rel})=\{(a_1, b_2)\}$

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$	$D_{1,1}=\{a_1,b_1\}$
$I_{1,1}(A_1(a_1))=\text{false}$	$I_{1,1}(B_1(b_1))=\text{true}$

Therefore, DISJOINT (A_i, B_j) ⊨ COR(A_i, B_j) does not hold.

A.3.6 Does COMPATIBLE(A_i, B_j) ⊨ DISJOINT(A_i, B_j) hold?

Does COMPATIBLE (A_i, B_j) ⊨ DISJOINT(A_i, B_j) hold?

$$\boldsymbol{F}_s(\exists x_i, y_j(\text{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))) \rightarrow$$
$$\boldsymbol{N}_{s'}(\forall x_i y_j(\text{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j)))) \wedge$$
$$\mathbf{G}_{s'}(\forall x_i y_j(\text{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j))))$$
$$\Rightarrow$$

$$\exists(st \in S)(L(s-1, st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))|_{z=st}] = 1) \rightarrow$$
$$(V_{(i,j),s'}[\forall x_i y_j(\mathrm{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j)))|_{z=s'}] = 1 \wedge$$
$$\forall(st' \in S)(L(s'-1, st') \rightarrow V_{(i,j),st'}[\forall x_i y_j(\mathrm{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j)))|_{z=st'}] = 1))$$
$$\Rightarrow$$
$$\exists(st \in S)(L(s-1, st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))|_{z=st}] = 1) \rightarrow$$
$$\forall(st' \in S)(L(s'-1, st') \rightarrow V_{(i,j),st'}[\forall x_i y_j(\mathrm{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j)))|_{z=st'}] = 1)$$
$$\Rightarrow$$
$$\exists(st \in S)(L(s-1, st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j)) \rightarrow$$
$$\forall x_i y_j(\mathrm{Rel}(x_i, y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j)))\,|_{z=st}]$$
$$\Rightarrow$$
$$\exists(st \in S)(L(s-1, st) \wedge V_{(i,j),st}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j) \rightarrow \neg(A_i(x_i) \wedge B_j(y_j))\,|_{z=st}])$$

This is *never* true, because it is false in every model.

Therefore, COMPATIBLE (A_i, B_j) ⊨ DISJOINT(A_i, B_j) does not hold, and moreover COMPATIBLE (A_i, B_j) always contradicts DISJOINT(A_i, B_j).

A.3.7 Does COMPATIBLE(A_i, B_j) ⊨Cor(A_i, B_j) hold?

Does COMPATIBLE(A_i, B_j) ⊨ COR(A_i, B_j) hold?

We only have to investigate the case when both relations are mappings.
This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=true
$L_{1,2}$={A_1,a_1}	$L_{2,2}$={B_2,b_2}
$D_{1,2}$={a_1}	$D_{2,2}$={b_2}
$I_{1,2}$(A_1(a_1))=true	$I_{2,2}$(B_2(b_2))=false

$I_{(1,2),1}$(Rel)={(a_1, b_2)}
$I_{(1,2),2}$(Rel)={(a_1, b_2)}

Therefore, COMPATIBLE(A_i, B_j) ⊨ COR(A_i, B_j) does not hold.

A.3.8 Does COR(A_i, B_j) ⊨Is(A_i, B_j) hold?

Does COR(A_i, B_j) ⊨ IS(A_i, B_j) hold?

If we use the same reasoning pattern as the first reasoning patterns in §0 and assume both sides are ontology mappings we will eventually get

$$\forall(st' \in S), L(s'-1, st') \to V_{(i,j),st'}[\forall x_i, y_j((A_i(x_i) \leftrightarrow B_j(y_j)) \to (\neg A_i(x_i) \vee B_j(y_j)))]$$

This is a *tautology* that is always true, because it is true for all models.

If we repeat the second reasoning pattern from §A.3.6, and assume that both relations are relations within an ontology (i.e. j=i), we will get:

$$\forall(st' \in S)(L(s'-1, st') \to V_{i,st'}[\forall x_i((A_i(x_i) \leftrightarrow B_i(x_i)) \to$$
$$((\neg A_i(x_i) \vee B_i(x_i)))\big|_{z=st'}] = 1)$$

This is a *tautology* that is always true.

Therefore, COR(A_i, B_j) ⊨ IS(A_i, B_j) is always true.

A.3.9 Does Is2(A_i, B_j) ⊨Is(A_i, B_j) hold?

Does IS2 (A_i, B_j) ⊨ IS(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=true	$I_{2,1}$(B_2(b_2))=false

$I_{(1,2),1}$(Rel)={(a_1, b_2)}

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}$={A_1,a_1, B_1,b_1}	$D_{1,1}$={a_1,b_1}
$I_{1,1}$(A_1(a_1))=true	$I_{1,1}$(B_1(b_1))=false

Therefore, $\text{Is2}(A_i, B_j) \models \text{Is}(A_i, B_j)$ does not hold.

A.3.10 Does $\text{Is}(A_i, B_j) \models \text{Is2}(A_i, B_j)$ hold?

Does $\text{Is}(A_i, B_j) \models \text{Is2}(A_i, B_j)$ hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))$=false	$I_{2,1}(B_2(b_2))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$

$D_{1,1}=\{a_1,b_1\}$

$I_{1,1}(A_1(a_1))$=false	$I_{1,1}(B_1(b_1))$=true

Therefore, $\text{Is}(A_i, B_j) \models \text{Is2}(A_i, B_j)$ does not hold.

A.3.11 Does $\text{Compatible}(A_i, B_j) \models \text{Is}(A_i, B_j)$ hold?

Does $\text{Compatible}(A_i, B_j) \models \text{Is}(A_i, B_j)$ hold?

We only have to investigate the case when both relations are mappings.
This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{1,2}=\{A_1,a_1\}$
$D_{1,1}=\{a_1\}$	$D_{1,2}=\{a_1\}$
$I_{1,1}(A_1(a_1))$=true	$I_{1,2}(A_1(a_1))$=true
$L_{2,1}=\{B_2,b_2\}$	$L_{2,2}=\{B_2,b_2\}$
$D_{2,1}=\{b_2\}$	$D_{2,2}=\{b_2\}$

$I_{2,1}(B_2(b_2))$=true $I_{2,2}(B_2(b_2))$=false

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$ $I_{(1,2),2}(Rel)=\{(a_1, b_2)\}$

Therefore, COMPATIBLE $(A_i, B_j) \vDash$ IS(A_i, B_j) does not hold.

A.3.12 Does DISJOINT$(A_i, B_j) \vDash$Is(A_i, B_j) hold?

Does DISJOINT$(A_i, B_j) \vDash$ IS(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=false

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$
$D_{1,1}=\{a_1,b_1\}$
$I_{1,1}(A_1(a_1))$=true $I_{1,1}(B_1(b_1))$=false

Therefore, DISJOINT$(A_i, B_j) \vDash$ IS(A_i, B_j) does not hold.

A.3.13 Does COR$(A_i, B_j) \vDash$Is2(A_i, B_j) hold?

Does COR$(A_i, B_j) \vDash$ IS2(A_i, B_j) hold?

If we use the same reasoning pattern as the first reasoning patterns in §A.3.6 and assume both sides are ontology mappings we will eventually get

$$\forall(st' \in S), L(s'-1, st') \to V_{(i,j),st'}[\forall x_i, y_j((A_i(x_i) \leftrightarrow B_j(y_j)) \to (A_i(x_i) \vee \neg B_j(y_j)))]$$

This is a *tautology* that is always true.

If we repeat the second reasoning pattern, and assume that both relations are relations within an ontology (i.e. j=i), we will get:

$$\forall(st' \in S)(L(s'-1, st') \to V_{i,st'}[\forall x_i((A_i(x_i) \leftrightarrow B_i(x_i)) \to (A_i(x_i) \vee \neg B_i(x_i)))\big|_{z=st'}] = 1)$$

This is a *tautology* that is always true.

Therefore, COR(A_i, B_j) ⊨ IS2(A_i, B_j) does always hold.

A.3.14 Does COMAPTIBLE(A_i, B_j) ⊨Is2(A_i, B_j) hold?

Does COMPATIBLE (A_i, B_j) ⊨ IS2(A_i, B_j) hold?

It does not hold, because here follows a counter-model.

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=false	$I_{2,1}$(B_2(b_2))=true
$L_{1,2}$={A_1,a_1}	$L_{2,2}$={B_2,b_2}
$D_{1,2}$={a_1}	$D_{2,2}$={b_2}
$I_{1,2}$(A_1(a_1))=true	$I_{2,2}$(B_2(b_2))=true

$I_{(1,2),1}$(Rel)={(a_1, b_2)}
$I_{(1,2),2}$(Rel)={(a_1, b_2)}

Therefore, COMPATIBLE (A_i, B_j) ⊨ IS2(A_i, B_j) does not hold.

A.3.15 Does DISJOINT(A_i, B_j) ⊨Is2(A_i, B_j) hold?

Does DISJOINT(A_i, B_j) ⊨ IS2(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$D_{1,1}$={a_1}	$I_{1,1}$(A_1(a_1))=false
$L_{2,1}$={B_2,b_2}	$D_{2,1}$={b_2}	$I_{2,1}$(B_2(b_2))=true

$I_{(1,2),1}$(Rel)={(a_1, b_2)}

The second case is when both are relations within the same ontology.
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$	$D_{1,1}=\{a_1,b_1\}$
$I_{1,1}(A_1(a_1))=\text{false}$	$I_{1,1}(B_1(b_1))=\text{true}$

Therefore, DISJOINT(A_i, B_j) ⊨ IS2(A_i, B_j) does not hold.

A.3.16 Does COR(A_i, B_j) ⊨Compatible(A_i, B_j) hold?

Does COR (A_i, B_j) ⊨COMPATIBLE(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))=\text{false}$	$I_{2,1}(B_2(b_2))=\text{false}$

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

Therefore, COR (A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) does not hold.

A.3.17 Does IS(A_i, B_j) ⊨Compatible(A_i, B_j) hold?

Does IS(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))=\text{false}$	$I_{2,1}(B_2(b_2))=\text{false}$

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

Therefore, IS(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) does not hold.

A.3.18 Does IS2(A_i, B_j) ⊨Compatible(A_i, B_j) hold?

Does IS2(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}$={A_1,a_1}	$L_{2,1}$={B_2,b_2}
$D_{1,1}$={a_1}	$D_{2,1}$={b_2}
$I_{1,1}$(A_1(a_1))=false	$I_{2,1}$(B_2(b_2))=false

$I_{(1,2),1}$(Rel)={(a_1, b_2)}

Therefore, IS2(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) does not hold.

A.3.19 Does DISJOINT(A_i, B_j) ⊨Compatible(A_i, B_j) hold?

Does DISJOINT(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) hold?

DISJOINT(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j)

$$\begin{aligned}
&\boldsymbol{N}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))) \wedge \\
&\boldsymbol{G}_s(\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))) \to \\
&\boldsymbol{F}_{s'}(\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))) \\
&\Rightarrow \\
&V_{(i,j),s}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))\big|_{z'=s}] = 1 \wedge \\
&\forall(st \in S)(L(s, st) \to V_{(i,j),st}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))\big|_{z'=st}] = 1)) \to \\
&\exists(st' \in S)(L(s'-1, st') \wedge V_{(i,j),st'}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st'}] = 1) \\
&\Rightarrow \\
&\forall(st \in S)(L(s-1, st) \to V_{(i,j),st}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j)))\big|_{z'=st}] = 1) \to \\
&\exists(st' \in S)(L(s'-1, st') \wedge V_{(i,j),st'}[\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st'}] = 1) \\
&\Rightarrow \\
&\exists(st' \in S)(L(s'-1, st') \wedge V_{(i,j),st'}[\forall x_i, y_j(\mathrm{Rel}(x_i, y_j) \to \neg(A_i(x_i) \wedge B_j(y_j))) \to \\
&\exists x_i, y_j(\mathrm{Rel}(x_i, y_j) \wedge A_i(x_i) \wedge B_j(y_j))\big|_{z'=st'}]
\end{aligned}$$

This *never* holds, because it is false in every model.

Therefore, DISJOINT(A_i, B_j) ⊨ COMPATIBLE(A_i, B_j) does not hold, and moreover DISJOINT(A_i, B_j) always contradicts COMPATIBLE(A_i, B_j).

A.3.20 Does IS2(A_i, B_j) ⊨Disjoint(A_i, B_j) hold?

Does IS2(A_i, B_j) ⊨ DISJOINT(A_i, B_j) hold?

This is not true, because here is a counter-model:

$L_{1,1}=\{A_1,a_1\}$	$L_{2,1}=\{B_2,b_2\}$
$D_{1,1}=\{a_1\}$	$D_{2,1}=\{b_2\}$
$I_{1,1}(A_1(a_1))$=true	$I_{2,1}(B_2(b_2))$=true

$I_{(1,2),1}(Rel)=\{(a_1, b_2)\}$

Does it hold inside a single ontology?
Again, in this situation the statement above is not true, because here is an example of a model where it is not satisfied (if we consider the semantics of these statements as defined in chapter 3):

$L_{1,1}=\{A_1,a_1, B_1,b_1\}$	$D_{1,1}=\{a_1,b_1\}$
$I_{1,1}(A_1(a_1))$=true	$I_{1,1}(B_1(b_1))$=true

Therefore, IS2(A_i, B_j) ⊨ DISJOINT(A_i, B_j) does not hold.