

# Multi-Context Reasoning in Continuous Data-Flow Environments

Von der Fakultät für Mathematik und Informatik  
der Universität Leipzig  
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM  
(Dr. rer. nat.)

im Fachgebiet

INFORMATIK

vorgelegt

von Dipl.-Ing. Stefan Ellmauthaler, BSc

geboren am 08. Mai 1986 in Wien.

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerhard Brewka, Universität Leipzig
2. Prof. Dr. Torsten Schaub, Universität Potsdam

Die Verleihung des akademischen Grades erfolgt mit Bestehen  
der Verteidigung am 07. Juni 2018 mit dem Gesamtprädikat magna cum laude.



---

## Acknowledgements

I thank Gerhard Brewka for being a very helpful thesis advisor and that he gave me the opportunity to move from Vienna to Leipzig for doing research within the scope of my thesis and the projects I am employed at. He always encourages the pursuance of own ideas and research directions, which get further improved by his good and constructive feedback and support. In addition, he was always caring for my well-being and has always been more dedicated than just being a supervisor and chief. Further thanks to Torsten Schaub for reviewing my thesis.

I am grateful for my colleagues which assemble a great cooperative community around the Intelligent Systems Group. They provide a good environment to exchange and discuss ideas, do research, and even communicating on a level where we talk about daily life and personal issues and achievements.

Another word of thanks shall be going to my co-authors from Vienna, Lisbon, and Leipzig who made incredible discussions and advances in the field of Argumentation and Multi-Context reasoning possible. Many parts of this work and my employment are funded and supported by the German Research Foundation (DFG) under grants BR – 1817/7 – 1/2 and FOR 1513. So thanks to the German Research Foundation, the Research Unit “1513 *Hybrid Reasoning for Intelligent Systems*”, and all the project partners and associates.

Additionally, I want to say “*thank you*” to all my friends who have spent free time together with me and who have given me more than one suggestion for concerns regarding the finishing of the doctoral study. Further thanks to my mother Ilse, who sadly died too early to see my whole development in Germany and the conclusion of my doctoral study, my father Erhard, and his second wife Brigitte for their support, love, caring and help. Last but not least I am very grateful for having my companion Mandy in my life. Her support and love is very important for me and being able to share a great deal of our time together is a priceless addition to my life.

Without all of you my road towards this thesis and the finishing of my doctoral studies would have been way rockier than it was.

*Thank you*

*Stefan*



---

## Abstract

In the field of artificial intelligence, research on knowledge representation and reasoning has originated a large variety of formats, languages, and formalisms. Over the decades many different tools emerged to use these underlying concepts and ideas. Each one has been designed with some specific application in mind and are even used nowadays, where the internet is seen as a service to be sufficient for the age of Industry 4.0 and the Internet of Things. In that vision of a connected world, with these many different formalisms and systems, it is imperative to have some unified formal way to exchange information, such as knowledge and belief. Alas, that alone is not enough, because even more systems get integrated into the online world and nowadays we are confronted with a huge amount of continuously flowing data. That means a solution is needed to both, allowing the integration of information and dynamic reaction to the data which is provided in such continuous data-flow environments.

This work is aiming to present a unique and novel pair of formalisms to tackle these two important needs and propose an abstract and general solution. We will introduce and discuss reactive Multi-Context Systems, which allow one to utilise different knowledge representation formalisms, so-called contexts which are represented as an abstract logic framework, and exchange their beliefs through the means of bridge rules with other contexts. These multiple contexts need to mutually agree on a common set of beliefs, an equilibrium of belief sets. While different Multi-Context Systems already exist, they are only solving this agreement problem once and are neither considering external data streams, nor are they reasoning continuously over time. reactive Multi-Context Systems will do this by adding means of reacting to input streams and allowing the bridge rules to reason with this new information. In addition we propose two different kind of bridge rules, declarative ones which are used to find a mutual agreement and operational ones to adapt the current knowledge for future computations.

The second framework is even more abstract and allows computations to happen in an asynchronous way. These asynchronous Multi-Context Systems are aimed at modelling and describing communication between contexts, with different levels of self-management and centralised management of communication and computation.

In this thesis, the reactive Multi-Context Systems, will be analysed with respect to usability, consistency management, and computational complexity, while we will show how asynchronous Multi-Context Systems can be used to capture the asynchronous ideas and how to model a reactive Multi-Context System with it. Finally we will show how reactive Multi-Context Systems are positioned in the current world of stream reasoning and that it can capture currently used technologies and therefore allows one to seamlessly connect different systems of these kinds with each other. Further on this also shows that reactive Multi-Context Systems are expressive enough to simulate the mechanics used by these systems to compute the corresponding results on its own as an alternative to the already existing ones.

---

For asynchronous Multi-Context Systems, we will discuss how to use them and that they are a very versatile tool to describe communication and asynchronous computation. In addition it will be shown that they can capture the notion of reactive Multi-Context Systems and therefore providing means to only synchronise groups of contexts while allowing the other contexts to operate asynchronously.

---

# Contents

---

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Motivation</b>	<b>5</b>
<b>3 Background</b>	<b>11</b>
3.1 Logics . . . . .	11
3.1.1 Classical Logics . . . . .	12
3.2 Complexity Theory . . . . .	14
3.3 Argumentation . . . . .	17
3.4 Multi-Context Systems . . . . .	21
<b>4 Knowledge Representation Formalisms</b>	<b>23</b>
4.1 Description Logics and Ontologies . . . . .	23
4.1.1 The Language $\mathcal{ALC}$ . . . . .	24
4.1.2 Reasoning with $\mathcal{ALC}$ . . . . .	26
4.1.3 Systems for Description Logic . . . . .	29
4.2 Answer Set Programming . . . . .	30
4.2.1 Logic Programs . . . . .	31
4.2.2 Algorithms for Answer Set Programming . . . . .	33
4.2.3 Answer Set Programming in Practice . . . . .	36
4.3 Abstract Dialectical Frameworks . . . . .	39
4.3.1 Syntax and Semantics . . . . .	41
4.3.2 Computational Aspects . . . . .	47
4.3.3 Systems for Abstract Dialectical Frameworks . . . . .	49
<b>5 Reactive Multi-Context Systems</b>	<b>53</b>
5.1 Original Reactive Multi-Context Systems . . . . .	54

## CONTENTS

---

5.2	Reactive Multi-Context Systems . . . . .	61
5.2.1	Components of a Reactive Multi-Context System . .	62
5.2.2	Semantics of Reactive Multi-Context System . . . .	65
5.2.3	Modelling with Reactive Multi-Context Systems . .	70
5.2.4	Inconsistency Management . . . . .	86
5.2.5	Limiting Non-Deterministic Effects . . . . .	94
5.2.6	Expressiveness . . . . .	95
5.2.7	Complexity . . . . .	98
<b>6</b>	<b>Asynchronous Multi-Context Systems</b>	<b>103</b>
6.1	Syntax and Semantics of Asynchronous Multi-Context Systems	104
6.2	Modelling an Example . . . . .	111
6.3	Simulating Reactive Multi-Context Systems . . . . .	115
6.4	Declarative Data Set Packing . . . . .	117
<b>7</b>	<b>Related Work</b>	<b>125</b>
7.1	Stream Reasoning Approaches . . . . .	126
7.2	Reactive Logic Programming . . . . .	134
7.3	Other Multi-Context Systems . . . . .	137
<b>8</b>	<b>Conclusion</b>	<b>141</b>
	<b>Bibliography</b>	<b>145</b>
	<b>Work and Author Information</b>	<b>157</b>
	Declaration of Authorship . . . . .	157
	Bibliographic Data . . . . .	158
	Academic History . . . . .	159
	Education . . . . .	159
	Awards . . . . .	159
	Academic Working Experience . . . . .	159
	Publications and Talks . . . . .	160



## Chapter 1

---

# Introduction

---

Research in the field of knowledge representation has originated a large variety of formats and languages. To use those formal concepts a wealth of tools have emerged (e.g. databases, ontologies, triple-stores, modal logics, temporal logics, nonmonotonic logics, logic programs under nonmonotonic answer set semantics, ...). Those tools were designed for specific needs of certain applications. With the idea of a “*connected world*”, nowadays we do not intend to divide information over different applications. It is desirable to have all information available for every application if need be. To express all of this knowledge, represented in specifically tailored languages, in a universal language would be too hard to achieve from the point of view of complexity as well as the troubles arising from the translation of the representations.

A second issue in current knowledge representation, which is already addressed in different fields of knowledge representation (e.g. stream data processing and querying [Zaniolo, 2012, Le-Phuoc et al., 2012], stream reasoning with answer set programming [Gebser et al., 2012a], forgetting in general [Lang and Marquis, 2010, Cheng et al., 2006]), is the lack of *online* usage of KR tools and formalisms. Most of the approaches only assume one-shot computations, which are triggered by a user. This may be a specific request in the form of a query to a computer. In practice there are many applications where knowledge is provided in a constant flow of information and it is desired to reason over this knowledge in a continuous manner. Such a constant flow of information can be seen as some dynamic environment, where our formalism should be able to deal with changed conditions and beliefs over time.

The concept of nonmonotonic Multi-Context Systems (MCS) [Brewka and Eiter, 2007] is a promising approach to achieve a formalism which will not suffer from any of the two shortcomings of current KR-languages. One context may be seen as an encapsulation of one knowledge representation formalism. There different knowledge formalisms may communicate by means of *bridge rules* to exchange beliefs with each other. Equilibria are used as semantics to ensure that none of the transferred beliefs may lead to changes in the

knowledge base and belief sets of the contexts which result in inconsistencies. The problem of connecting divided knowledge was the motivation of MCS and its successor [Brewka et al., 2011b]. To generalise the concept of MCS such that they can handle *streams of information* and *react* to those in an appropriate way is one of the fundamental problems which shall be addressed by this thesis.

We aim to define and introduce so called *reactive Multi-Context Systems*, picturing how they developed over the last few years. Together with the presentation of this formalism we are also considering how to use this system and show results on computational complexity, ways to simulate other existing systems, and properties with respect to inconsistency management. Then we will propose and discuss another framework, *asynchronous Multi-Context Systems*, which use a more loosely coupled semantics compared to reactive Multi-Context Systems. While one basic idea of a reactive Multi-Context System is to get an agreement of every context via the usage of *equilibrium* semantics, asynchronous Multi-Context Systems are tailored towards information spreading in an asynchronous way, similar to the current web-technologies. We will discuss how asynchronous Multi-Context Systems can be seen as a modelling tool to describe communication and information exchange in modern KR and AI systems. In addition we will show that reactive Multi-Context Systems may be simulated with asynchronous Multi-Context Systems too, showing that this formalism can indeed utilise the equilibrium semantics where it is needed.

The contributions of this thesis can be summarised as follows:

- Introduction of reactive Multi-Context Systems
  - Two extensions of managed Multi-Context Systems, to model the necessary concepts relevant to deal with dynamic environments
  - An introduction of another type of rules, which allows the separation between equilibria computation and dynamic changes of knowledge over time
  - A study of different forms of inconsistencies, together with plans to ensure repairs or non-existence of inconsistent non-computable paths
  - Ways to model other reactive systems from the literature with reactive Multi-Context Systems
- Introduction of asynchronous Multi-Context Systems
  - The introduction of a paradigm shift regarding bridge rules (now called output rules)
  - An alternative model of computation, where each context produces its beliefs on its own pace
  - A language to pre-filter relevant from irrelevant data

- 
- Methods to work on partial results and control the flow of computation based on these observations

This thesis is structured as follows. Chapter 2 will present our real world related examples, which will be used through the introduction and introduction of the contributions of this work.

All related and needed Background for later chapters will be presented in chapter 3. The revisited topics in the Background section will first introduce a general view on logics and show on basis of classical propositional logic how it can be used. Afterwards we will refresh the knowledge about complexity theory, which allows to classify computational problems based on their worst case runtime. Another basic knowledge representation and reasoning formalism, the argumentation theory will then be presented. The chapter will be concluded by the formal introduction of managed Multi-Context Systems.

A short overview on different knowledge representation formalisms which are nowadays used in business and research will be shown in chapter 4. There we will focus on three concepts, ranging from monotone to nonmonotone reasoning semantics. One important factor in this chapter is that each of the pictured approaches have implementations and use-able software applications.

Then we will introduce reactive Multi-Context Systems in chapter 5. To show how it has evolved we will have a discussion on the different versions which got proposed till the final, most usable, and best analysed version will be presented. There we will also discuss their properties and how to tackle inconsistencies introduced by bridge rules.

Afterwards Chapter 6 will present asynchronous Multi-Context Systems and discuss their relation to reactive Multi-Context Systems. In Chapter 7 we discuss related work. Finally Chapter 8 will conclude the thesis and give an outlook on future work as well.

The contributions of this work are based on the following publications: Section 4.3 has been discussed in a handbook on argumentation<sup>1</sup> and has been accompanied by a Journal publication in the *IfCoLog Journal of Logics and their Applications* [Brewka et al., 2017b]. Beside that many parts have been proposed by various conference papers [Ellmauthaler and Strass, 2016, Ellmauthaler and Strass, 2014, Brewka et al., 2013, Ellmauthaler and Wallner, 2012] and different workshop and system description submissions [Ellmauthaler and Strass, 2013, Strass and Ellmauthaler, 2017]. Note that this work is also based on the work of my master's thesis [Ellmauthaler, 2012]. Chapter 5 presents the results of the recently published *AIJ (Artificial Intelligence Journal)* article [Brewka et al., 2018]. This work has been discussed before in different conference publications [Brewka et al., 2016a, Brewka et al., 2014b], as well as in workshop contributions [Brewka et al., 2016b, Brewka et al., 2014a]. Finally, Chapter 6's work is based on two workshop papers [Ellmauthaler and Pührer,

---

<sup>1</sup>The “*Handbook of Formal Argumentation*” (*HOFA*) is going to appear, further details can be found at <http://formalargumentation.org/>

2014, Ellmauthaler and Pührer, 2016] and a contribution in the *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation* Festschrift, which has been published to celebrate the 60<sup>th</sup> birthday of Gerhard Brewka [Ellmauthaler and Pührer, 2015].

## Motivation

---

Nowadays our world is getting more connected and is filled with machine-readable and semantically aware data. *Internet of Things* (IoT) [Want et al., 2015, Atzori et al., 2010, Weiser, 1991] is one example which gets more important and present in the daily life. There each thing in the real world gets its own identification on the internet and may provide information about itself as well as offering ways to manipulate their states. Together with this new level of interconnections *Industry 4.0* [Salkin et al., 2018, Bauer et al., 2016] gets more prominent in business applications too. Industry 4.0 is a keyword for enhancing the production and sales processes to utilise the current and future possibilities of the connected virtual world. Another example is the *Semantic Web* as a way to describe relations between information in the web and annotate it with classifications and semantics to allow automatic reasoning on the content of information. Beside the industry, medicine is aware of the possible gain of quality and security by using the new technologies for their use cases too.

Eventually we can easily say that there is already a vast amount of machine-processable information and knowledge available. This is a preparatory ground for a new class of dynamic, rich, and knowledge-intensive applications, which connect heterogeneous knowledge bases.

In most of these new applications, it is still the case that each component does its own kind of computation without being aware of other observations or conclusions. That is due to independent, sometimes overlapping, development of languages and formalisms. They are not necessarily compatible to each other and therefore their knowledge representation, reasoning, set of beliefs, and other concepts may differ on a level where a direct integration is not practicable. Information and therefore knowledge and beliefs are just exchanged as data and it is often the case that each unit of the whole network is working and reasoning on its own. In case of more complex reasoning units, which utilise different knowledge representation and reasoning formalisms, this may lead to inconsistent or not even realisable situations.

To overcome these communication issues Multi-Context Systems [Brewka

et al., 2011a] have been developed. Such a system consists of various Contexts, where each Context is some kind of reasoning engine (e.g. Answer Set Programs, Argumentation Systems, Description Logic Bases, First Order Reasoner, . . .). They provide a common formalism to exchange and integrate knowledge and beliefs over different heterogeneous sources, such as different reasoner engines. A way to ensure that this communication does not lead to inconsistencies due to the integration of external sources is a strong semantics which ensures that each part of the system agrees on the information which is transferred between the contexts. This mutual agreement is gained by achieving an equilibrium based on the transferred information. Later on that concept got enhanced even more. Managed Multi-Context Systems [Brewka et al., 2011b] are not only adequate to exchanging, and therefore adding, information. In addition to adding information, they are capable of revising the knowledge bases of the receiving context. That means that based on some external information, knowledge might be added, removed or changed.

Many knowledge representation and reasoning systems follow the paradigm of static knowledge based reasoning. Managed Multi-Context Systems are no exception of that. In the real world we are situated in an environment which confronts every individual thing continuously with new influences and changing situations. Everything adapts to these external and even internal changes over time which leads to ongoing dynamics. These dynamics can be seen as a constant adjustment and evolving of parts of the environment to fit the current situations. Of course it would be beneficial if our knowledge based system which integrates information from different sources is also aware of this continuous change of states. To observe the world, one is confronted with a constant flow of information.

There are some examples of systems developed with the purpose of reacting to streams of incoming information, such as Reactive ASP [Gebser et al., 2011a, Gebser et al., 2012a], C-SPARQL [Barbieri et al., 2010], Ontology Streams [Lécué and Pan, 2013] and ETALIS [Anicic et al., 2012], to name only a few. However, they are very limited in the kind of knowledge that can be represented, and the kind of reasoning allowed, hence unsuitable to address the need to integrate heterogeneous knowledge bases.

Our aim is to propose a system which might shift from the static knowledge based approach to a point where it react and adapt over time to the continuous data-flow of its environment. We will achieve this by generalising the managed Multi-Context Systems approach to get a reactive formalism which is aware of the dynamics and still allows the integration of other sources of knowledge. As we are challenged with a stream of information about current observations, it won't be sufficient to only compute one equilibrium. So we will compute a sequence of equilibria, where one equilibrium is based on its predecessor and its current observations and influences from its environment. This will enable one to present a logically traceable and formally clear way to incorporate the environmental properties to the computing contexts and track the adaption

---

of knowledge and beliefs of the system over time. Inconsistencies which might occur during such adaptations will need to be detected and we will need to find ways to repair or avoid such events, as this might render the logical sound approach to an halt. We will present such a generalised system, which we call *reactive Multi-Context System* and investigate how to handle such inconsistent cases.

While consistency is ensured by building an equilibrium, this means that for the computation of such a result every context needs to finish its computations and conclusions too. As every context needs to wait for all other contexts to finish its reasoning, we will call this a *strongly coupled semantics* because the semantics couple the contexts together and reduce their freedom of computation. In contrast to that procedure we are interested in *loosely coupled semantics* which allow more freedom in between the contexts. We are still interested in a way to exchange information between different contexts. Obviously we want some methods to get consistent behaviour and similar properties as the equilibrium based approach combined with the possibilities to reason on partial results of contexts. In case some computation may take way too long for the computation of an agreement there should be some ways to control the computation process from outside the context too. These ideas are the conceptual basic thoughts of *asynchronous Multi-Context System*. An asynchronous Multi-Context System should be capable of simulating the strong coupling and computation of equilibria, while it should also allow decoupled computations and reactions to streams of input data as it is handled nowadays in the internet.

To illustrate how such systems might be used and how these are utilised for usage in the real world, we will consider two different running examples. Both of them are situated on possible real word applications, where computation of reasonable solutions with robust conclusions and without inconsistencies is necessary. To underline that importance we are choosing two applications which are related to health-care. We are taking two different sets of examples to underline once the importance of the strongly coupled semantics on one hand, while we are also on the aspect of the necessity of fast interaction too. In the latter case we are even considering problems where a consensus of all contexts would not be very reasonable. Both example scenarios will use state of the art knowledge representation and reasoning engines, to show how current technology might be used in such use-case. For better readability and easier understanding of the newly introduced concepts we will also use tuned down and simplified versions of current databases and data structures<sup>2</sup>. Note that we are aware that such an application would need long medical quality studies and may also be opposed by the question of responsibility or trust into computer

---

<sup>2</sup>Examples for such down-tuning are the description logic based ontologies which are huge topologies for biomedical facts from the real world or argumentation systems which might become very complex solutions to instantiate and propagate relations between arguments.

based decisions in such delicate environments. Our incentive is to propose possible applications and introduce ways to model interaction between different knowledge based reasoning systems with a strong and robust formal basis.

### Example scenarios

**Assisted Living** In this scenario we are considering some use-case from the concept of assisted living. This is the idea to aid some person with assistance in daily life to overcome some (in most cases medical) shortcoming. Such treatment is often done proposed for people who suffer different physical or mental restrictions. In our special case, we will consider John<sup>3</sup>, an elderly person suffering from dementia. People with such a brain disease are not faring well with big changes of their living environment, so it is beneficial to allow one to stay at home and get assistance as long as possible before being forced into some retirement and treatment home. Our idea is that John is living alone in his apartment which is equipped with various sensors, e.g. smoke detectors, cameras, state detectors (e.g. is the stove on, is the light turned on, etc.), and body sensors (e.g. pulse, blood pressure, etc.).

An assisted living application in such a scenario could leverage the information continuously received from the sensors, together with John's medical records stored in a relational database, a biomedical health ontology with information about diseases, their symptoms and treatments, represented in some description logic, some action policy rules represented as a non-monotonic logic program, to name only a few, and use it to detect relevant events, suggest appropriate action, and even raise alarms, while keeping a history of relevant events and John's medical records up to date, thus allowing him to live on his own despite his condition. After detecting that John left the room while preparing a meal, the system could alert him in case he does not return soon, or even turn the stove off in case it detects that John fell asleep, not wanting to wake him up because his current treatment/health status values rest over immediate nutrition. Naturally, if John is not gone long enough, and no sensor shows any potential problems (smoke, gas, fire, etc.), then the system should seamlessly take no action.

Another application might be the adaption of reasoning about John's medical status, beside deciding whether some action should be concluded or not. Lets assume the sensors report that John has tachycardia (i.e. a very fast heart rate). The system checks the medical report of John, together with the history of his taken drugs from an automated drug dispenser. Based on the information and reasoning from the biomedical health ontology the system concludes that the tachycardia is due to the recently taken decongestant drugs. Due to the current policy rules and the high possibility that this is just a side effect of John's medication, the system update the medical report on John to

---

<sup>3</sup>John as in John Doe, to ensure that no one might be offended by the choice of the given name.



---

add this knowledge, together with a note about the policy rules limits. On one hand it is reasonable to log such actions for control of the system and on the other hand it is easier to reason at the next time instant whether an action should be taken because of the reoccurring event of tachycardia at the next sensor cycle.

Note that we are not proposing a system where a computer does critical decisions on medical cases. We are envisioning and describing an assisting embedded solution to help making it easier to handle daily life. Actions to be done are in most situations either an alert for John or a supervising medical employee or some danger avoiding action like turning off the stove.

**Computer-Aided Emergency Team Management** Now we want to describe a scenario which has more focus on direct interaction between entities of the real world and the system which provides the reasoning tasks. This application is a recommender-system for the coordination and handling of ambulance assignments. We consider the scenario, where some person is calling the emergency number. During the rescue call many information regarding the casualty is given and the emergency response employee needs to assess the situation as quickly as possible. When the call is handled the information is usually forwarded to some case dispatcher, which plans how to utilise the accessible ambulance cars in the most effective way. The employee who does this dispatching has to take many parameters into account, like free capacities in hospitals, ambulance car availability and equipment, traffic, severity of the individual cases, and much more. These employees have to react to additional random events like a broken ambulance car, accidents on the streets, and similar issues which may change previously totally acceptable preferences for the task assignments.

Such a computer-aided emergency team management application can be seen as one system which can deal with the dynamics we have described above while it also incorporates the integration of information gathered from the different employees. The emergency response employee can interact with the system by seeing his input as a flow of sensor data and the computed conclusions can be presented as part of the reasoning of the system. Information from the call provided by the employee is directly given to a case-analyser, which starts to annotate the case based on the given data. It might also pass some of its conclusions on to a medical ontology context which checks the consistency of the given information and provides additional assessment of severity of the case. These conclusions are presented to the employee who can decide if the proposed annotation is acceptable or if he wants to add or remove some of the estimation. As soon as the case is rated it can be handed on to a task planner context (e.g. some ASP planner) which computes optimal assignments for the ambulance car. This context will take additional information into account, like navigational data for estimated time of arrivals, real-time traffic ratings, or

additional annotations from the medical ontology. Then it interacts with the case dispatching employee and presents its optimal solution candidates. The dispatcher might now choose its own solution, recycles some of the candidates or just chooses one of these. Based on the interaction with the human employee the system now might reassess its policies and rules for the decision making, but more important it will inform the assigned ambulances about there cases too, causing updates in the context for keeping track of the available ambulances.

Note that this example is tailored towards direct interaction and fast reaction over time, underlining how the information flow of the sensors is causing reassessment and additional information and knowledge flow inside the system. Again we want to strongly emphasise that we are aiming on a computer-aided solution which might learn from user interaction for a speed up in decision making.

---

# Background

---

The following sections will give a short introduction to basic background knowledge which will be required in further chapters of this work. Here we will present mature concepts, formalisms, and ideas already introduced and discussed by other authors.

## 3.1 Logics

Studies about Logics can be traced back to Greek philosophers like Aristotle and Plato. It is an extensively analysed and discussed field of science and got even more interest from the mathematical point of view since the early twentieth century. Therefore a plethora of different approaches, formalisms and methods emerged during the investigation of that field <sup>4</sup>. We will use a rather generalised and abstract view on logics, as it has been used as 'formal concept of logic' in [Brewka and Eiter, 2007]:

**Definition 3.1.1** (Logic). *A logic is a triple  $L = \langle KB, BS, \mathbf{acc} \rangle$ , where*

- *KB is a set of knowledge bases,*
- *BS is a set of belief sets, and*
- *$\mathbf{acc} : KB \mapsto 2^{BS}$ , the acceptance function is a function which assigns to each knowledge base a set of belief sets.*

Intuitively the set of knowledge bases for the logic may be seen as the syntax of the described formalism. It defines which combination of knowledge may be assumed as a working base. In that manner the acceptance function can be seen as the semantics, as it defines which beliefs or conclusions shall be drawn based on the given knowledge base.

---

<sup>4</sup>We want to point the interested reader towards the comprehensive series *handbook of philosophical logic* [Gabbay and Guenther, 2014] for a detailed overview on the landscape of existing logics.

This notion of a logic can represent different forms of logics. The biggest advantage of that formalisation is, that it may model all common types of logics, while it can also model other computational models and different problems which may arise from handling formal logics.

In general we can say that  $L$  is *monotonic* if **acc** assigns to each  $kb \in KB$  a single belief set  $B \in BS$ , and  $kb \subseteq kb'$  of another  $kb' \in KB$  implies that  $B \subseteq B'$ . Moreover it is possible to have an acceptance function which has more than one belief set as well as no result for a given knowledge base, which leads to the possibility to also model a *nonmonotonic* logic. In the remainder of this section we will show how this concept is instantiated with a basic classical logic and how to model different formal problems and solutions with that abstract notion of a logic..

### 3.1.1 Classical Logics

Classical propositional logic is a basic and well understood two-valued monotone logic. For further information and a more exhaustive in-depth introduction we would like to refer the interested reader towards handbooks about logics [Gabbay et al., 1993, Church, 1996, Rothmaler, 2000, Huth and Ryan, 2004].

As a binary logic every variable can only have one of two values (i.e. a variable is either true or false). A *well-formed formula* is a syntactical valid formulation of a proposition, consisting of a combination of variables, constants and operations.

**Definition 3.1.2** (Well-formed Propositional Formula). *Given a signature  $\Sigma := (\Sigma_c, \Sigma_{pv}, \Sigma_{op})$ , where  $\Sigma_c = \{\top, \perp\}$  is a set of constant symbols,  $\Sigma_{pv}$  is a set of propositional variables, and  $\Sigma_{op} = \{\neg, \vee, \wedge, \supset\}$ <sup>5</sup>. is a set of operations, a well-formed propositional formula is inductively defined as:*

- (i) Every  $p \in \Sigma_{pv}$  is a formula.
- (ii) Every  $p \in \Sigma_c$  is a formula.
- (iii) If  $\phi$  is a formula, then  $(\phi)$  is a formula too.
- (iv) If  $\phi$  is a formula, then  $\neg\phi$  is a formula too.
- (v) If  $\phi$  and  $\psi$  are formulae and  $\circ \in (\Sigma_{op} \setminus \{\neg\})$ , then  $\phi \circ \psi$  is one too.

Formulae defined in (i) and (ii) are called *atomic formulae*. Atoms together with their negation ( $\neg$ ) are referred to as *literals*.

To give the well-formed formulae a meaning we are going to define the *semantics of propositional logic*. First we need to identify the truth value of a given formula. For such a computation it is needed to map the used variables towards a truth value each. This mapping is called an *interpretation*.

---

<sup>5</sup>Note: It is just for convenience to use more than two operators for the semantics

**Definition 3.1.3** (Interpretation of a Formula). *An interpretation  $I : \Sigma_{pv} \mapsto \{0, 1\}$  is a total function, which assigns to each variable a value.*

Based on the given interpretation for a given formula it is now possible to compute the *valuation* of the formula. The valuation of the formula can be seen as the truth value of the given formula with respect to the given interpretation.

**Definition 3.1.4** (Valuation of an Formula). *Let  $\Sigma$  be a signature,  $\phi, \psi$  be two arbitrary formulae under  $\Sigma$ , and  $I$  an interpretation of  $\Sigma$ , then*

- (i)  $v_I(p) = I(p), p \in \Sigma_{pv}$ ,
- (ii)  $v_I(\top) = 1$  and  $v_I(\perp) = 0$ ,
- (iii)  $v_I(\neg\phi) = 1 - v_I(\phi)$ ,
- (iv)  $v_I(\phi \wedge \psi) = \min(v_I(\phi), v_I(\psi))$ ,  
 $v_I(\phi \vee \psi) = \max(v_I(\phi), v_I(\psi))$ , and  
 $v_I(\phi \supset \psi) = \max(1 - v_I(\phi), v_I(\psi))$ .

We say that an interpretation is a model of a given formula, if the valuation under the interpretation is true. Furthermore a formula is satisfiable if there exists at least one model for it. A valid formula is a formula where every interpretation is a model. Two formulae are equivalent if their models are equivalent too. In addition we are interested in the logical consequences of a given set of formulae. In other words, what does also hold if some formulae are given to be holding?

**Definition 3.1.5** (Entailment). *Let  $\Gamma$  be a set of formulae under a signature  $\Sigma$  and  $\phi$  be a formula under  $\Sigma$ .*

$$\Gamma \models \phi \text{ iff } \forall(I)(\forall(\psi \in \Gamma)v_I(\psi) = 1) \rightarrow v_I(\phi) = 1$$

Intuitively that means, that some formula  $\phi$  is entailed by a set of formulae  $\Gamma$ , if every interpretation which is a model for each formula in  $\Gamma$  is also a model for  $\phi$ . The logical consequence of some propositions (i.e. a set of propositional formulae) is the set of all formulae which are entailed by the propositions. As there are many different representations of the same formula, we are only interested in the set of formulae which are unique with respect to their equality.

To consider propositional logic for the abstract notion of logic, we say that  $F_\Sigma$  is the set of all well-defined formulae constructed by  $\Sigma$ . We will write  $L_p = \langle KB_p, BS_p, \mathbf{acc}_p \rangle$  to denote that this logic representation is for propositional logic. All the admissible knowledge bases are given by the set  $KB_p = 2^F$  and the belief sets  $BS_p$  is the set of all deductively closed sets of formulae over  $\Sigma$ .  $\mathbf{acc}_p$  now maps to every  $kb \in KB_p$  to one  $B \in BS$ , such that each  $B$  is the corresponding logical consequence to  $kb$ . It is also easy to model some of the above mentioned problems, though they are not really

logical consequences. One could use  $L_{pSAT} = \langle KB_{pSAT}, BS_{pSAT}, \mathbf{acc}_{pSAT} \rangle$ , where  $KB_{pSAT} = F$  and  $BS_{pSAT} = \{\{0\}, \{1\}\}$  to encapsulate the SAT-problem.  $\mathbf{acc}_{pSAT}$  is then mapping each formula  $\phi \in KB_{pSAT}$  to true or false, depending on whether  $\phi$  is satisfiable or not.

## 3.2 Complexity Theory

Computational complexity is aimed towards the analysis of the hardness of problems and formalisms. The theory gives means to compare and classify computational problems by the basic idea whether some problem may be solved by another formalism. The idea of complexity classes is that every problem of one class may be transformed in polynomial time into a problem of the same class such that the answer of the transformed problem is dual to the answer of the original problem. In this section we give an overview on complexity theory and the common complexity classes. A brief overview is given by Johnson [Johnson, 1992] and for an in-depth insight in complexity theory we refer to the book by Papadimitriou [Papadimitriou, 1994].

A problem in complexity theory is defined by an input description and a question to be answered. We will speak of a *decision problem* if the question requires to answer either “yes” or “no”. To assess the complexity we are now interested in a function which only depends on the input and the method to solve the problem. A *complexity class* is a set of functions which has a similar magnitude in space or time growth, based in the input size.

An easy way to define decision problems is to use an *universal Turing machine*.

**Definition 3.2.1** (Turing Machine). *A TM is a tuple  $\langle Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F \rangle$ , where*

- $Q \subseteq \mathbb{Q}$  is a finite, non-empty set of states,
- $\Gamma \subseteq \mathbb{S}$  is a finite, non-empty set of tape symbols, the alphabet,
- $\sqcup \in \Gamma$  is the blank symbol,
- $\Sigma$  is a sequence of alphabet symbols, the input,
- $q_0$  is the initial state,
- $F \subseteq Q$  is the set of final states, and
- $\delta : Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$  is the (partial) transition function.

A Turing machine is literally a tape where the input is written on. Based on the current state and the alphabet symbol on the current position on the tape the transition function may change the state, the symbol on the tape as

well as the position on the tape. Note that the position may only be one step to the left or the right on the tape which is represented by  $\leftarrow$  and  $\rightarrow$  respectively.

We will speak of a *universal Turing machine*  $U = \langle TM \rangle$  if and only if  $U$  takes any turing machine as input and has the same answer as the specified turing machine will give. A turing machine is deterministic if the transition function is deterministic as well. We speak of a deterministic universal Turing machine if the turing machine is a deterministic one.

**Definition 3.2.2** (Complexity Class  $\mathbf{P}$ ). *A problem  $P$  is in  $\mathbf{P}$  if it can be solved by a deterministic universal turing machine in polynomial many working steps, with respect to the length of the input string.*

In other words a problem in  $\mathbf{P}$  needs, based on the input length, a polynomial factor of time to compute a solution. One example for a problem in  $\mathbf{P}$  is the decision problem whether an interpretation  $I$  for a formula  $\phi$  is a propositional model  $I \in \text{mod}_p(\phi)$  or not.

A non-deterministic Turing machine has a non-deterministic transition function, i.e. one value might map to more than one result, where it is not defined which one will be used.

**Definition 3.2.3** (Complexity Class  $\mathbf{NP}$ ). *A problem  $P$  is in  $\mathbf{NP}$  if it can be solved by a non-deterministic universal turing machine in polynomial many working steps, with respect to the length of the input string.*

To define  $\mathbf{NP}$  we used a non-deterministic universal turing machine. That means that the transition function is non-deterministic. Alas, with nowadays means of technology it is only possible to construct a deterministic turing machine. Intuitively we can see a non-deterministic turing machine as a deterministic turing machine which is guessing the “right” value of the possible ones to produce a computation path which leads to a result. Alternatively it can be seen as a deterministic turing machine, which tries out every computational path within one computation step. An example for a problem in the  $\mathbf{NP}$  class is the SAT problem for propositional formulae (i.e. is a given formula  $\phi$  satisfiable or not). Note that it is unknown if there exists an efficient way to compute problems which are in  $\mathbf{NP}$ , but current solutions to solve  $\mathbf{NP}$  problems with deterministic methods take exponential time in the worst case.

We will say a problem is  *$\mathbf{NP}$ -complete* if we know that the problem has a *membership* in this class and that the problem is  *$\mathbf{NP}$ -hard*.

**Definition 3.2.4** (Membership and Hardness). *A problem  $P$  has a membership in a complexity class  $C$ , if an algorithm exists, whose complexity function is in the class of  $C$ .*

*A problem  $P$  is said to be  $C$ -hard for a complexity-class  $C$ , if a program  $\Pi$  exists, which transforms  $P'$  to  $P$ , where  $P'$  is known to be a  $C$ -hard problem, and the answer to  $P'$  equals the answer to  $P$ . Additionally the complexity of  $\Pi$  must not be greater than  $\mathbf{P}$ .*

One method to show the **NP**-membership is to use a “guess & check” algorithm. This algorithm will guess a solution and afterwards checks whether the solution is correct or not. Note that the algorithm only checks one guess. If this “guess & check” algorithm has a polynomial runtime (i.e. has a **P** membership), then the problem is in **NP**.

For each non-deterministic problem class a *complement class* exists (e.g. **coNP**). There all answers are the complement of the original problem. One example of two complement problems is the SAT and the UNSAT problem. Note that the two complement problems can have a different difficulty to solve: To answer the SAT question, it is only needed to test the interpretations till one is a model, but to check for the UNSAT answer every interpretation must be tested to show that no interpretation is a model.

On top of the classes **P**, **NP**, and **coNP**, we can now define one additional type of classes. We will use so-called *oracles*. Let us assume we have an oracle which can solve a problem in a complexity class with a constant computational effort of one unit of time. If we use such an oracle in our program the overall complexity of the program without the oracle would be higher than with the oracle. In case we have an program in **P** and an oracle which solves a problem in **NP**, we would have the complexity class **P<sup>NP</sup>**. Based on this notation for algorithms with oracles we can build the *polynomial hierarchy*.

**Definition 3.2.5** (Polynomial Hierarchy).

$$\begin{aligned}\Delta_0^{\mathbf{P}} &= \Sigma_0^{\mathbf{P}} = \Pi_0^{\mathbf{P}} = \mathbf{P}; \\ \text{and for all } i \geq 0 : \\ \Delta_{i+1}^{\mathbf{P}} &= \mathbf{P}^{\Sigma_i^{\mathbf{P}}} \\ \Sigma_{i+1}^{\mathbf{P}} &= \mathbf{NP}^{\Sigma_i^{\mathbf{P}}} \\ \Pi_{i+1}^{\mathbf{P}} &= \mathbf{coNP}^{\Sigma_i^{\mathbf{P}}}\end{aligned}$$

To get a better understanding on the hierarchy, Figure 3.1 is sketching the relations of the different classes. Note that we have already mentioned exponential time complexity **EXPTIME**, which is defined like **P**. We had only a look on computation time so far. It is easy to extend the notion by measuring the needed size of the turing machine’s tape. In a similar way we can define the notion of **PSPACE** such that the space requirements on the input tape are a polynomial factor of the input size. In addition a less detailed notion for computational complexity exists for computational problems. Here the problems are assigned to be *tractable* or *intractable*. In general all problems in **P** are said to be *tractable* problems while all problems which are in a higher complexity class are *intractable* computational problems.

Note that it is currently an open problem whether the polynomial hierarchy is a correct assumption or not. Indeed the question whether  $\mathbf{P} \neq \mathbf{NP}$  holds or not is still not answered. Nevertheless we will assume that  $\mathbf{P} \subseteq \mathbf{NP}$  and



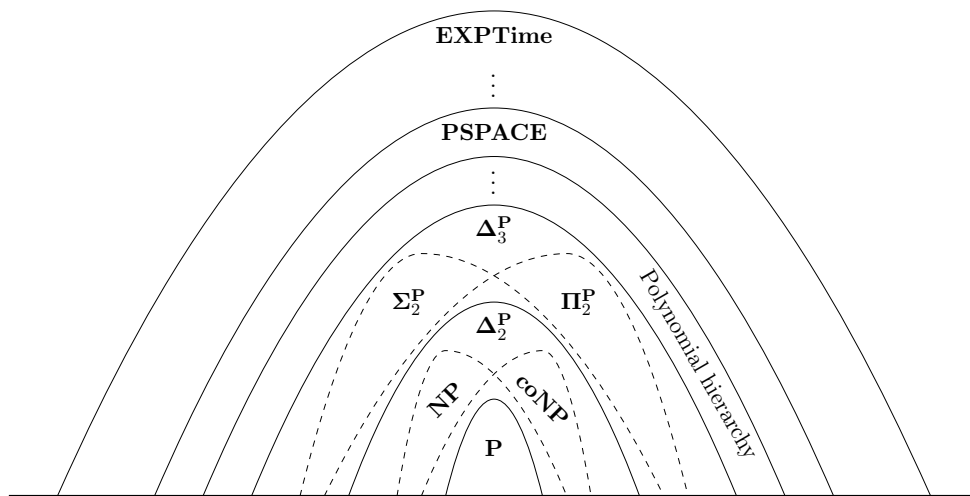


Figure 3.1: Relations of the classes in the polynomial hierarchy

$NP \not\subseteq P$ . Following this assumption we will also stick to the notion of the polynomial hierarchy as presented, which is the current point of view of the scientific community.

### 3.3 Argumentation

Argumentation is an area at the intersection of Philosophy, Knowledge Representation, nonmonotonic Reasoning, and Multi-Agent Systems, which receives growing interest over the last decade. It is based around the idea to construct, evaluate, and position arguments to model the (nonmonotonic) reasoning which may arise during having an argument or discussion. Note the contrast towards proof theoretic approaches, where everything which has been proven has to stay admissible. In argumentation theory every argument is defeasible, which means that their conclusion may change due to the influence of another argument. As a result of such an argumentation some admissible set of arguments should be found, which stand together as some conclusion which is not changed any more by the other present arguments.

Formal argumentation can be seen as a *three step process* [Caminada and Wu, 2011], which is tailored towards the very abstract concept of Argumentation Frameworks (see Figure 3.2). Normally one has some kind of knowledge base and a given problem to solve on top on it. The *instantiation* process takes the knowledge base and produces different arguments. Note that arguments may be very abstract and there is no necessity to keep any internal structure for arguments. Then the arguments are set into relations to each other (e.g. 'does

one argument is in conflict with another one', 'does one argument support another one', ...). With such a set of arguments and their positions to each other constructed, we get an *argumentation framework*. In the next step, a set of acceptable arguments is identified via different semantics which are defined for the given framework. Such acceptable sets are in general called *extensions*. In the last step, *conclusions are identified*. They are in a direct relation to the given extensions and can be considered as the wanted result of the original problem originated from the knowledge base.

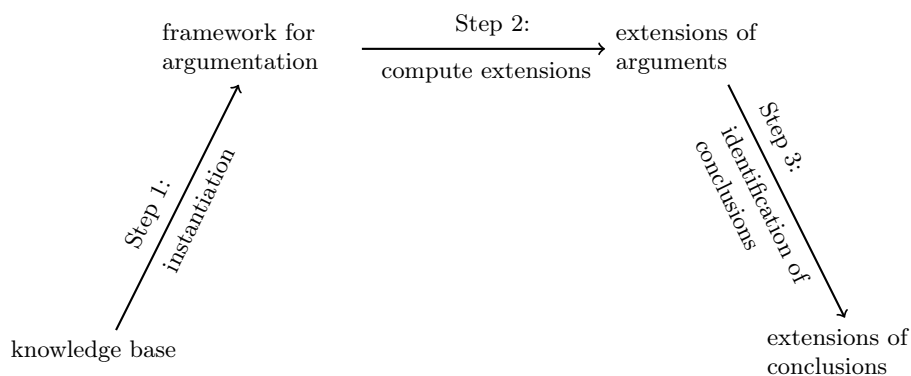


Figure 3.2: Three-step argumentation process

In *abstract argumentation* we are only interested in the second step, namely the computation of extensions and the representation of arguments with their relations. One very simple, but still widely used and accepted argumentation framework is *Dung's Argumentation Framework* (Dung AF) [Dung, 1995]. While the approach is pretty simple, it is still a nonmonotonic reasoning formalism. Intuitively every argument is seen very abstract. There is no more information than that it is an argument and which other argument is attacked by it. Attack is meant that there is a strong reason that the other argument shall not be acceptable if the attacking argument is accepted too. Note that attacks are not mutual and so this notion is directional.

**Definition 3.3.1** (Dung's Argumentation Framework). *A Dung Argumentation Framework  $AF = \langle Arg, Att \rangle$  is a tuple, where*

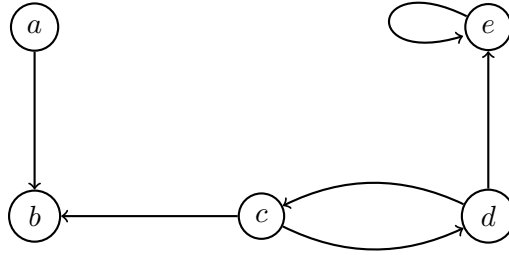
- *Arg is a set of arguments and*
- *Att  $\subseteq Arg \times Arg$  is a binary relation*

The intended meaning of  $(a, b) \in Att$  for two arguments  $a, b \in Arg$ , is that  $a$  attacks  $b$ . It is trivial to see that the definition of a Dung AF is identical to the definition of a directed graph. Therefore it is common to represent such a

framework as a graph for easier visualisation, where arguments are the vertices and the attack relations are the edges.

**Example 3.3.2.**

The  $AF_1 = \langle \{a, b, c, d, e\}, \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\} \rangle$  can be easily represented by the following graph:



In that example the argument  $a$  is not attacked by any other argument and has a point against  $b$ , which is also attacked by  $c$ . In addition  $c$  and  $d$  are attacking each other mutually.  $r$  is not only attacked by  $d$ , but is also attacked by itself. That may result due to some inconsistency behind  $e$ .

Note that we will use  $AF_1$  in further examples to show how the different semantics for Dung Argumentation Frameworks work.

Dung has defined several semantics for his framework, which are all considered as “basic semantics”. Their connection between each other is well investigated. Basically the semantics, so called *Extensions*, are acceptable sets of arguments with respect to their relations between them. The most basic notion which is also mandatory for most following semantics is *conflict-freeness*, which means that no two arguments are allowed to be selected together if there is an attack-relation between them.

**Definition 3.3.3** (Conflict-free). *Let  $AF = \langle Arg, Att \rangle$  be a Dung Argumentation Framework. A set  $S \subseteq Arg$  is called conflict-free if there are no two arguments  $a, b \in S$ , such that  $(a, b) \in Att$ . The set  $cf(AF)$  denotes the set of all conflict-free sets for  $AF$ .*

In general one is not only interested in some kind of consistency between the selected arguments, as it is achieved by conflict-freeness. We are also taking into account those arguments which are not in the extension. Literally it is our goal that arguments may also defend against other attacks, by providing an attack towards the attacker itself. In other words we want to rebut all arguments which are attacking those which are considered as acceptable.

**Definition 3.3.4** (Defended Arguments). *Let  $AF = \langle Arg, Att \rangle$  be a Dung Argumentation Framework. An argument  $a \in Arg$  is said to be defended by a set  $S \subseteq Arg$ , iff for each argument  $b \in Arg$  : if  $(b, a) \in Att$ , then there exists*

a  $c \in S$ , such that  $(c, b) \in \text{Att}$ . We will write  $\text{def}(AF, S)$  for the set of all arguments which are defended by  $S$  in  $AF$ .

**Definition 3.3.5** (Extensions for Dung Argumentation Frameworks). *Let  $AF\langle \text{Arg}, \text{Att} \rangle$  be a Dung Argumentation Framework. A conflict-free set  $S \in \text{cf}(AF)$  is called*

- *admissible if  $S \subseteq \text{def}(AF, S)$ ,*
- *complete if  $S = \text{def}(AF, S)$ ,*
- *preferred if  $S$  is a maximal admissible extension with respect to set-inclusion,*
- *grounded if  $S$  is the least complete extension with respect to set-inclusion, and*
- *stable if for every  $a \in \text{Arg} \setminus S$  exists a  $s \in S$ , such that  $(s, a) \in \text{Att}$ .*

Intuitively the different extensions can be interpreted as follows. An admissible extension needs to be able to defend itself against all other attacks from other arguments, while the complete extension requires that every defended argument inside the set too. The preferred extension is only interested in the maximal admissible sets which sets the focus on those sets which have the most arguments. In contrast to that the grounded extension is aiming towards all sets which can be selected without doubt. Stable extensions take some kind of the “safest way”, such that every argument which is not part of their extension needs to be attacked by at least one argument from the extension.

Dung has also presented different properties for the extensions. For every Dung AF there always exists an admissible, complete, preferred, and grounded extension. In addition the grounded extension is always a unique set. The sets of extensions are also in a special relation to each other, such that *stable*  $\subseteq$  *preferred*  $\subseteq$  *complete*  $\subseteq$  *admissible*. Note that the grounded extension itself is also a subset of every stable, preferred and complete extension.

**Example 3.3.6.** *Assume  $AF_1$  from Example 3.3.2. Then the extensions are as follows:*

- *admissible:  $\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}$*
- *complete:  $\{a\}, \{a, c\}, \{a, d\}$*
- *preferred:  $\{a, c\}, \{a, d\}$*
- *grounded:  $\{a\}$*
- *stable:  $\{a, d\}$*

### 3.4 Multi-Context Systems

Here we will present Multi-Context Systems [Brewka and Eiter, 2007, Brewka et al., 2011a], more specifically their extended form which is called managed Multi-Context Systems [Brewka et al., 2011b]. Multi-Context Systems provide means to exchange beliefs between different knowledge representation formalisms. These different formalisms are encapsulated in so-called *contexts*. To represent the flow of information (i.e. beliefs) between contexts, each context has rules to define when actions need to be performed. One important point of this concept is to provide strong semantics to only allow exchange of information which does not result in inconsistencies for the individual context. The basic version of Multi-Context Systems only allow addition of information in the various contexts, while the managed version introduces the freedom to use different operators to modify the underlying knowledge bases and semantics of the contexts. That may be just addition or removal of distinct parts of the knowledge base and may go as far as revision or enforcement of different beliefs. Note that Multi-Context Systems use the notion of logics, as detailed in Section 3.1, while managed Multi-Context Systems use *logic suites*, a more flexible and abstracted version. These suites allow the definition of different semantics for one logic where later one is chosen by the context.

**Definition 3.4.1** (Logic Suite). *A logic suite  $LS = (KB_{LS}, BS_{LS}, ACC_{LS})$  consists of the set  $BS_{LS}$  of possible belief sets, the set  $KB_{LS}$  of well-formed knowledge-bases, and a nonempty set  $ACC_{LS}$  of possible semantics of  $LS$ , i.e.  $\mathbf{acc}_{LS} \in ACC_{LS}$  implies  $\mathbf{acc}_{LS} : KB_{LS} \rightarrow 2^{BS_{LS}}$ .*

Each logic suite defines a set of formulae  $F_{LS} = \{s \in kb \mid kb \in KB_{LS}\}$  which does represent all formulae which may occur in any knowledge base of  $LS$ . To make the changes in the knowledge base, some operations need to be defined. The set of allowed operations is handled through a *management base*  $OP$ . For one logic suite  $LS$  and a management base  $OP$ , the set of *operational statements* which can be built from  $OP$  and  $F_{LS}$  is  $F_{LS}^{OP} = \{o(s) \mid o \in OP, s \in F_{LS}\}$ . Next the semantics for each statement in  $F_{LS}^{OP}$  is specified via the *management function*.

**Definition 3.4.2** (Management Function). *A management function over a logic suite  $LS$  and a management base  $OP$  is a function  $mng : 2^{F_{LS}^{OP}} \times KB_{LS} \rightarrow 2^{KB_{LS} \times ACC_{LS}} \setminus \{\emptyset\}$ .*

This function allows to modify the given knowledge base in any way which results in another valid knowledge base. In addition it chooses one semantics  $\mathbf{acc} \in ACC$  which should be used. Note that this management function is not a deterministic one.

**Definition 3.4.3** (Managed Multi-Context System). *A managed Multi-Context System  $M$  is a collection  $(C_1, \dots, C_n)$  of managed contexts where, for  $1 \leq i \leq n$ , each managed context  $C_i$  is a quintuple  $C_i = (LS_i, kb_i, br_i, OP_i, mng_i)$  such that*

- $LS_i = (BS_{LS_i}, KB_{LS_i}, ACC_{LS_i})$  is a logic suite,
- $kb_i \in KB_{LS_i}$  is a knowledge base,
- $OP_i$  is a management base,
- $br_i$  is a set of bridge rules for  $C_i$ , with the form

$$op_i \leftarrow (c_1 : p_1), \dots, (c_j : p_j), not(c_{j+1} : p_{j+1}), \dots, not(c_m : p_m).$$

such that  $op_i \in F_{LS_i}^{OP_i}$  and for all  $1 \leq k \leq m$  there exists a context  $c_k \in (C_1, \dots, C_n)$  such that  $p_k \in S \in BS_{LS_{c_k}}$ , and

- $mng_i$  is a management function over  $LS_i$  and  $OP_i$ .

The introduced bridge rules are handling the exchange of information between different contexts. Based on the (non) existence of beliefs of any context (even the context itself) one or more operational statements may become applicable and get executed by the management function associated to the context. For one bridge rule  $r \in br_i$  the notion of  $op(r)$  denotes the operator  $op_i \in F_{LS_i}^{OP_i}$ , while  $body(r)$  is the set  $\{(c_{k_1} : p_{k_1}) \mid 1 \leq k_1 \leq j\} \cup \{not(c_{k_2} : p_{k_2}) \mid j < k_2 \leq m\}$  which represents the part right of the  $\leftarrow$  sign of the rule.

How the applicable operators for the management function are selected and whether the changes to the different contexts is still consistent is determined by the *equilibrium semantics*. A belief state  $\mathbf{B} = \langle b_1, \dots, b_n \rangle$  for a given managed Multi-Context System is a tuple which holds one belief set for each context. Furthermore we use the set of applicable operators  $app_i(\mathbf{B}) = \{op(r) \mid r \in br_i \wedge \mathbf{B} \models body(r)\}$  to designate all operators for one context which can be applied with respect to the given belief state.

**Definition 3.4.4** (Equilibria for managed Multi-Context Systems). *Let  $M = (C_1, \dots, C_n)$  be a managed multi-context system. A belief state  $\mathbf{B} = \langle b_1, \dots, b_n \rangle$  is an equilibrium of  $M$  iff for every  $1 \leq i \leq n$  there exists some  $(kb'_i, \mathbf{acc}_{LS_i}) \in mng_i(app_i(S), kb_i)$  such that  $S_i \in \mathbf{acc}_{LS_i}(kb'_i)$ .*

Intuitively the computation of an equilibrium is in such a way that first one belief is guessed. Based on this guess the applicable operational statements are determined and computed by the management function for each context. In case the belief state is now acceptable by the modified knowledge bases under the semantics chosen by the management function, then the belief state is an equilibrium.

---

# Knowledge Representation Formalisms

---

We have introduced Multi-Context Systems in Section 3.4. We are interested in modelling Multi-Context reasoning where our examples are apprehended from health care applications, therefore this chapter will facilitate on the introduction of modern and nowadays used knowledge representation formalisms. The choice for introducing these formalisms in favour against others is that each one has systems which can be used to solve given reasoning tasks within the formalism. In addition they are all based on methods and concepts which have been accepted either in leading artificial intelligence conferences or from industry. They are also ranging from monotonic, complex reasoning tasks (i.e. Description Logics 4.1) to nonmonotonic reasoning (i.e. Answer Set Programming 4.2 and Abstract Dialectical Frameworks 4.3). In the following chapters we will use these concepts and formalisms in the ongoing examples of assisted living and the emergency case handling application. We will analyse and present each formalism in a formal manner first. Then we will give a short overview on existing systems and general usability in the real world.

## 4.1 Description Logics and Ontologies

Description Logic [Baader, 2003] is a family of different attributive languages, where the basic version is called  $\mathcal{AL}$  (*attributive language*). This family of languages is designed to represent knowledge in a monotonic way. Like the name suggests, it captures the terminology of different *concepts* and their *roles* to describe (or classify) concepts and individuals. Description logic uses expressions from First Order Logic (or Predicate Logic; for more details see the logic handbook [Huth and Ryan, 2004]) to form statements more expressive than those propositional logic is capable of. While Predicate Logic is a powerful and expressive tool, it has the shortcoming of being undecidable. This does not hold for Description Logic because it restricts the expressiveness of the

language, such that Description Logic is a decidable fragment of First Order Logic. A specific collection of terminologies to some topic is called an *ontology*. The following introduction will follow the Description Logic Handbook, using some terminologies from the logic handbook.

#### 4.1.1 The Language $\mathcal{ALC}$

Intuitively the language  $\mathcal{AL}$  consists of Concepts which can be seen as unary Predicates in Predicate Logic, and Roles which are binary relations between two Concepts. In addition knowledge is represented in two separated spaces. The *TBox* consists of all definitions of atomic Concepts, Concepts and Role names, while the *ABox* contains all assertions which connect individuals to the different Concepts and sets them in relation to each other.

**Definition 4.1.1** (Concept Descriptions for  $\mathcal{AL}$ ).

$C, D \rightarrow$	$A \mid$	<i>(atomic concept)</i>
	$\top \mid$	<i>(universal concept)</i>
	$\perp \mid$	<i>(bottom concept)</i>
	$\neg A \mid$	<i>(atomic negation)</i>
	$C \sqcap D \mid$	<i>(intersection)</i>
	$\forall R.C \mid$	<i>(value restriction)</i>
	$\exists R.\top \mid$	<i>(limited existential quantification)</i>

With the basic concept it is possible to already describe some basic ideas. For example if *Person* and *Male* are a primitive concept, it is possible to describe the concept which defines persons which are not male by  $Person \sqcap \neg Male$ . In addition with the atomic role *isFriendof* it is possible to have concepts like  $Person \sqcap \exists isFriendof.\top$  or  $Person \sqcap \forall isFriendof.Male$  which capture all persons who have friends or those who have exclusively male friends respectively. To define a formal semantics for the already intuitively described concepts we will use the notion of an interpretation  $\mathcal{I}$ .

**Definition 4.1.2** (Semantics of  $\mathcal{AL}$  Concepts). *The Interpretation  $\mathcal{I}$  is a tuple  $\langle \Delta^{\mathcal{I}}, val \rangle$ , where  $\Delta^{\mathcal{I}}$  is a nonempty set which is called the domain of the interpretation.  $val$  is a valuation function which assigns to every atomic Concept  $A$  a set  $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$  and to each atomic Role  $R$  a binary relation  $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ .*



For extended Concept descriptions the valuation function is defined as follows:

$$\begin{aligned}
 \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
 \perp^{\mathcal{I}} &= \emptyset \\
 (\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
 (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
 \forall R.C^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall (b).(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
 \exists R.\top^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists (b).(a, b) \in R^{\mathcal{I}}\}
 \end{aligned}$$

Equivalence between concepts is defined over semantics in the usual manner, such that two concepts  $C$  and  $D$  are equivalent if and only if  $C^{\mathcal{I}} = D^{\mathcal{I}}$  for all interpretations  $\mathcal{I}$ .

One big advantage of the family of Description Logic is the option to extend the language by different features. In general every feature is related to some character of the alphabet and to denote that change the character is added to the name of the language. One example might be the permission to also negate arbitrary concepts. The language is then called  $\mathcal{ALC}$  with  $\mathcal{C}$  for Compliments. For such an extension the valuation function needs also to be extended by

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},$$

where  $C$  is some arbitrary concept.

Note that there are some extensions which do not change the expressiveness of the language, but can be added as syntactic sugar for convenience. Two most common used are the *union of concepts* and the *full existential qualifier*, denoted by  $\mathcal{U}$  and  $\mathcal{E}$  respectively. Of course the union might be achieved by simply applying De Morgan's laws<sup>6</sup> and the full existential qualifier is gained by utilising the duality of existential and universal quantifiers<sup>7</sup>. To still have a sound valuation function the two definitions need to be added accordingly, such that

$$\begin{aligned}
 (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \text{ and} \\
 \exists R.C^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists (b).(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.
 \end{aligned}$$

In general the most basically used Description Logic is  $\mathcal{ALC}$ <sup>8</sup>, where it is also common to include  $\mathcal{E}$  and  $\mathcal{U}$  without further addressing these redundant convenience additions further.

With syntax and semantics on arbitrary Concepts and Roles at hand, we can now define a so-called *Terminology*. It allows to relate new Concepts and

<sup>6</sup>i.e.  $A \sqcap B \equiv \neg(\neg A \sqcup \neg B)$

<sup>7</sup>i.e.  $\exists R.C \equiv \neg \forall R.\neg C$

<sup>8</sup>For that reason we will stick to  $\mathcal{ALC}$  now, unless it is expressed differently

Roles to others by relating them to each other. Informally spoken, it is a way to say that one Concept (or Role) is another Concept(Role) too, either reflexive (i.e. equality) or irreflexive (i.e. inclusions). We will write

$$C \sqsubseteq D \quad (\text{respectively } R \sqsubseteq S)$$

$$C \equiv D \quad (\text{respectively } R \equiv S)$$

to denote that a Concept  $C$  is included by  $D$ , a Role  $R$  is included by  $S$  or that they are equivalent respectively. A Terminology is a finite set of definitions (i.e. equalities and inclusions). Due to its nature of being a collection of definitions such a Terminology is also called a *TBox*  $\mathcal{T}$ . We will say that  $\mathcal{I}$  satisfies the inclusion if

$$C \sqsubseteq D \text{ if } C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$$

and that  $\mathcal{I}$  satisfies the equivalence if

$$C \equiv D \text{ if } C^{\mathcal{I}} = D^{\mathcal{I}}$$

hold. In addition we say that  $\mathcal{I}$  is a model of  $\mathcal{T}$  if  $\mathcal{I}$  satisfies every definition in  $\mathcal{T}$ .

The *TBox* describes how different kinds of concepts and roles relate to each other. It is obvious that these Terminologies do not contain any information about real world entities. The description of the world itself is done in the *ABox*. Inside such an *ABox* are assertions about individuals to connect them to the terminologies. We will denote individuals with lower letters. To express that an individual  $a$  is part of a Concept  $C$  we write  $C(a)$  and to denote that the individuals  $a, b$  are in a Role  $R$  we use  $R(a, b)$  respectively. Like a *TBox* needs to be finite, also the *ABox* is a finite set of assertions. To include the *ABox* into the semantics, we need to extend the definition of the interpretation  $\mathcal{I}$  such that  $\mathcal{I} = \{\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\}$ , where  $\Delta^{\mathcal{I}}$  remains unchanged and  $\cdot^{\mathcal{I}}$  maps each individual name  $a$  to one element  $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ . Note that in Description Logics the *Unique Name Assumption* holds, which means that two different names  $a$  and  $b$  need to be different elements in the Interpretation (i.e.  $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ ). An interpretation  $\mathcal{I}$  satisfies an assertion  $C(a)$  if  $a^{\mathcal{I}} \in C^{\mathcal{I}}$  and it satisfies a Role  $R(a, b)$  if  $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ . In addition an interpretation satisfies an *ABox*  $\mathcal{A}$  if it satisfies every assertion in  $\mathcal{A}$ . Finally, the interpretation  $\mathcal{I}$  satisfies the *ABox*  $\mathcal{A}$  with respect to the *TBox*  $\mathcal{T}$  if  $\mathcal{I}$  satisfies  $\mathcal{A}$  and  $\mathcal{I}$  is a model of  $\mathcal{T}$ .

### 4.1.2 Reasoning with $\mathcal{ALC}$

In the previous section we have defined how an interpretation  $\mathcal{I}$  satisfies an *ABox* with respect to a *TBox*. Now we will have a closer look on different ways for inference and reasoning for the family of Description Logics. Again we will stick to  $\mathcal{ALC}$  as it is one very prominent language in this family. First we will recap about different inference types on a *TBox* and how to reduce all

of these problems into unsatisfiability questions, then we will present a tableau algorithm to check satisfiability.

We can consider the following inference tasks for a *TBox*:

**Satisfiability** A concept  $C$  is satisfiable with respect to  $\mathcal{T}$  if there exists a model  $\mathcal{I}$  of  $\mathcal{T}$ , such that  $C^{\mathcal{I}}$  is nonempty. That is when  $\mathcal{I}$  is a model of  $C$ .

**Subsumption** A concept  $C$  is subsumed by a concept  $D$  with respect to  $\mathcal{T}$  if  $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$  for every model  $\mathcal{I}$  of  $\mathcal{T}$ . We will write  $\mathcal{T} \models C \sqsubseteq D$  or  $C \sqsubseteq_{\mathcal{T}} D$ .

**Equivalence** A concept  $C$  is equivalent to a concept  $D$  with respect to  $\mathcal{T}$  if  $C^{\mathcal{I}} = D^{\mathcal{I}}$  for every model  $\mathcal{I}$  of  $\mathcal{T}$ . We will write  $\mathcal{T} \models C \equiv D$  or  $C \equiv_{\mathcal{T}} D$ .

**Disjointness** Two concepts  $C$  and  $D$  are disjoint with respect to  $\mathcal{T}$  if  $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$  for every model  $\mathcal{I}$  of  $\mathcal{T}$ .

**Proposition 4.1.3** (Reduction to Unsatisfiability [Baader, 2003]). *Let  $C$  and  $D$  be two Concepts, then*

- (i)  $C$  subsumes  $D \iff C \sqcap \neg D$  is unsatisfiable,
- (ii)  $C$  is equivalent to  $D \iff C \sqcap \neg D$  and  $\neg D \sqcap C$  are unsatisfiable, and
- (iii)  $C$  is disjoint to  $D \iff C \sqcap D$  is unsatisfiable.

*This also holds with respect to a terminology  $\mathcal{T}$ .*

Based on these transformations it is now possible to formulate each of these inference types as an unsatisfiability question. This is a necessary preparation, as the tableau algorithm we will present can answer the question of satisfiability (and henceforth unsatisfiability) for a given *ABox*. Due to the latter restriction of that algorithm we now have to transform the *TBox* into an *ABox*, which is done with the *TBox-elimination*. This technique instantiates  $\mathcal{T}$ , such that individuals are connected to each concept and role. In this way an *ABox*  $\mathcal{A}$  is constructed. The tableau algorithm is a method to generate a set of *ABoxes* which are consistent to the  $\mathcal{T}$ , such that there exists an interpretation which satisfies the *ABox* with respect to the *TBox*.

In the following we will expand the notion of an instance of a concept (or role), such that  $C(a)$  still means that  $a$  is an instance of  $C$  but an arbitrary  $\mathcal{ALC}$  formula  $F(a)$  means that the individual  $a$  is an individual which occur in  $F$ . Intuitively it can be seen as a filler notion till all concepts and roles are instantiated to an actual assertion. We call this step to transform a *TBox* into an *ABox* a *TBox-elimination*. The tableau algorithm starts with an  $\mathcal{ALC}$

formula in negated normal form<sup>9</sup> with an eliminated *TBox*. This eliminated *TBox* is then added to an existing *ABox* which may be empty though. So we get that  $F(a) \in \mathcal{A}$ . Starting with this *ABox*, a set of Rules is then applied one after another, till no further rule can be applied or an inconsistency appears in the *ABox*. Each rule adds more information to the *ABox* and may also split the *ABox* into two alternatives. Note that the algorithm may produce finitely many *ABoxes*. If one *ABox* is inconsistent this one is said to have a *clash*.

**Definition 4.1.4** (Tableau Algorithm Rules).

- $\rightarrow \sqcap$ -rule  
**Requires:**  $\mathcal{A}$  contains  $(C \sqcap D)(a)$ , but both  $C(a)$  and  $D(a)$  are not in  $\mathcal{A}$ .  
**Action:**  $\mathcal{A}' := \mathcal{A} \cup \{C(a), D(a)\}$ .
- $\rightarrow \sqcup$ -rule  
**Requires:**  $\mathcal{A}$  contains  $(C \sqcup D)(a)$ , but neither  $C(a)$  nor  $D(a)$  is in  $\mathcal{A}$ .  
**Action:**  $\mathcal{A}' := \mathcal{A} \cup \{C(a)\}, \mathcal{A}'' := \mathcal{A} \cup \{D(a)\}$ .
- $\rightarrow \exists$ -rule  
**Requires:**  $\mathcal{A}$  contains  $(\exists R.C)(a)$ , but no  $b$  exists, such that  $C(b)$  and  $R(a, b)$  are in  $\mathcal{A}$ .  
**Action:**  $\mathcal{A}' := \mathcal{A} \cup \{C(b), R(a, b)\}$ , where  $b$  does not occur in  $\mathcal{A}$  as an individual.
- $\rightarrow \forall$ -rule  
**Requires:**  $\mathcal{A}$  contains  $(\forall R.C)(a)$  and  $R(a, b)$ , but no  $C(b)$ .  
**Action:**  $\mathcal{A}' := \mathcal{A} \cup \{C(b)\}$ .

If every *ABox* has a clash, then the *ABox* with respect to the original *TBox* is inconsistent. Every *ABox* which has no clash and no further rule can be applied is a consistent *ABox* with respect to the *TBox*. In the following Example, we will show how this tableau is applied to a small terminology.

**Example 4.1.5.** Assume a terminology  $\mathcal{T}$  where we have the primitive concepts  $S$  which means that  $a$  is a student and the primitive role  $K$  which describes that a student  $a$  knows another student  $b$ . In addition the terminology includes the following definitions

$$\begin{aligned} S1 &\equiv S \sqcap \exists K. \neg S \text{ and} \\ S2 &\equiv S \sqcap \exists K. \forall K. \neg S. \end{aligned}$$

Now we want to prove or reject the following statement:

$$\mathcal{T} \models S1 \sqsubseteq S2$$

---

<sup>9</sup>Negated normal form can be constructed as it is usual for propositional or predicate formulae. Note that a transformation into negated normal form is always terminating and every formula has a negated normal form.

First we reduce the subsumption into an unsatisfiability problem, such that  $S1 \sqsubseteq \neg S2$  should be unsatisfiable with respect to  $\mathcal{T}$ . Then we use TBox elimination on the formula, to gain  $(S1 \sqsubseteq \neg S2)(a)$ . Next we expand the definitions into the formula:

$$(S \sqcap \exists K. \neg S \sqcap \neg(S \sqcap \exists K. \forall K. \neg S))(a)$$

To apply the tableau algorithm we now need to transform it into negated normal form:

$$(S \sqcap \exists K. \neg S \sqcap (\neg S \sqcup \forall K. \exists K. S))(a)$$

Finally we can apply the tableaux algorithm.

1.	$(S \sqcap \exists K. \neg S \sqcap (\neg S \sqcup \forall K. \exists K. S))(a)$	Query in NNF
2.	$(S \sqcap \exists K. \neg S)(a)$	$\rightarrow \sqcap(1)$
3.	$(\neg S \sqcup \forall K. \exists K. S)(a)$	$\rightarrow \sqcap(1)$
4.	$S(a)$	$\rightarrow \sqcap(2)$
5.	$(\exists K. \neg S)(a)$	$\rightarrow \sqcap(2)$
$\swarrow \quad \searrow$		
6.	$\neg S(a) \quad (\forall K. \exists K. S)(a)$	$\rightarrow \sqcup(3)$
7.	$\otimes \quad K(a, b)$	$\rightarrow \exists(5)$
8.	$\neg S(b)$	$\rightarrow \exists(5)$
9.	$(\exists K. S)(b)$	$\rightarrow \forall(6)$
10.	$K(b, c)$	$\rightarrow \exists(9)$
11.	$S(c)$	$\rightarrow \exists(9)$

Figure 4.1: Tableau algorithm for Example 4.1.5

The algorithm ends with one clash and one clash-free ABox. Therefore the formula is satisfiable which means that the subsumption does not hold. Note that the clash-free ABox also provides a counter model, which shows that the subsumption does not hold. The ABox

$$\mathcal{A} = \{S(a), \neg S(b), S(c), K(a, b), K(b, c)\}$$

does satisfy  $S1$ , as  $a$  is a student and  $a$  knows  $b$  who is not a student. Furthermore it does not satisfy  $S2$ , as  $a$  is a student, who knows some person  $b$ , but  $b$  does not exclusively know persons who are no students, as  $b$  knows  $c$ , which is a student too.

### 4.1.3 Systems for Description Logic

The Description Logic formalism is generally used for biological, medical, and biomedical applications (see overview articles like [Hoehndorf et al., 2015]). Due to this pretty broad usage of this language, a standard for ontologies in

the web has been worked out by the *W3C* consortium. The *Ontology Web Language* (OWL) [W3C OWL Working Group, 2009] is commonly used for constructing ontologies and it has different kind of subclasses which are related to the different sets of Description language families. They conceptualise and classify real world knowledge about cells, molecules, drugs, health-care, and similar structures. One of the most prominent ontologies in medicine is *SNOMED* [Cornet and de Keizer, 2008]<sup>10</sup>, which classifies drugs, disorders, side-effects, synergies, diseases, and anatomy in one big set of terminologies.

To automatically solve queries given to an ontology there are also different solvers available. It seems that each of them has due to different algorithms used in the implementation their special fields where one outperforms the other. One comparison of three very prominent solvers can be found in [Motik et al., 2009], where they also introduce the HERMIT solver. The other two established and used solvers are PELLET [Sirin et al., 2007] and FACT++ [Tsarkov and Horrocks, 2006]. Note that *OWL* and therefore also these solvers operate on more expressive description logics than *ALC*.

## 4.2 Answer Set Programming

Answer set programming is a logic programming formalism, which has some common roots with logic programming languages like *PROLOG* [Colmerauer et al., 1973]. On the topic of logic programming in general we would like to point the interested reader to Bob Kowalski's overview article on the beginnings of logic programming [Kowalski, 1988].

The basic idea of answer set programming is to produce stable models which are grounded and closed under the given program, while in *PROLOG* a deduction of given rules is done to answer a query or produce some interpretation<sup>11</sup>. In the following we will first present answer set programming. First we will have a look on the monotonic version of *definite logic programs* and afterwards we will extend that notion to gain the nonmonotonic *normal logic programs* which are the basic formalism of answer set programming. There are excellent books about answer set programming, like Gelfond and Kahls book on *KR* with answer set programming as an approach [Gelfond and Kahl, 2014] or *Answer Set Solving in Practice* by the Group of the Potsdam University [Gebser et al., 2012b]. A shorter insight to the topic can be done by having a look at one well written overview journal article [Brewka et al., 2011c]. That introduction will be followed by a short section about algorithms to compute models of answer set programs. Finally we will describe some modelling techniques and conclude with the discussion about some solvers on that topic.

---

<sup>10</sup>Note that this is an overview article on over forty years of research on that topic

<sup>11</sup>That also means that cycles may result in endless deductions in *PROLOG*, which is no problem for answer set programming.

### 4.2.1 Logic Programs

A normal logic program is a finite set of rules. Each rule can be seen as an implication, which means that if the premise holds, then the conclusion should hold too.

**Definition 4.2.1** (Definite Logic Program Rule). *A definite logic program rule  $r$  is of the form*

$$a \leftarrow b_1, \dots, b_n$$

where  $a$  as well as  $b_1$  to  $b_n$  are ground atoms.  $a$  is called the head of the rule, denoted by  $hd(r)$  and  $b_1, \dots, b_n$  is the body of the rule, denoted by  $body(r)$ . If the body is empty, we call the rule a fact. In this case it is conventional to omit the  $\leftarrow$  symbol too.

The result of such a program is the set of atoms which are the consequences of all of its rules. Of course we want that each atom from the rulehead of every rule whose body is fulfilled will be part of the consequence of the program, while we are not interested in any atoms, which are not resulting from any of these ruleheads.

**Definition 4.2.2** (Consequence of a Definite Logic Program). *Let  $S$  be a set of atoms and  $P$  be a definite logic program.*

- *$S$  is closed under  $P$  if and only if  $a \in S$  whenever  $a \leftarrow b_1, \dots, b_n \in P$  and  $\{b_1, \dots, b_n\} \subseteq S$ .*
- *Let  $R = (r_1, \dots, r_n)$  be a finite sequence of rules of  $P$ , such that each atom in the body of a rule  $r_i$  is a head of rule  $r_j$ , where  $j < i$ . We call  $R$  a derivation of  $P$  and  $atoms_d = \{hd(r) \mid r \in R\}$  is the set of all atoms derived by  $R$ .*
- *$S$  is grounded in  $P$  if and only if  $a \in S$  implies that  $a \in atoms_d(R)$  holds in one derivation of  $P$ .*

*The consequence  $Cn(P)$  of  $P$  is a set, which is closed under and grounded in  $P$ .*

Note that this definition of the consequence of a program is pretty straightforward and easy to grasp. It is also reasonable to rewrite each implication in a disjunction. The resulting set of horn formulas then has one minimal model, which also happens to be a unique one. That one model corresponds to the consequence of the program.

Obviously, there are only deterministic choices made, as everything which can be directly deduced by the rules will be in the result. In the bodies of rules are only positive atoms allowed, so the head will be added to the answer if there is already a proof available for every atom in the body. It is not possible to express that a rule body should be satisfied if there is no deduction for some

atom. To allow this, we introduce the nonmonotonic (or weak) negation. If an atom in the body is weakly negated, that means that there is no derivation for that atom so far.

**Definition 4.2.3** (Normal Logic Program Rule). *A normal logic program rule  $r$  is of the form*

$$a \leftarrow b_1, \dots, b_n, \text{not } c_1, \dots, \text{not } c_m$$

where  $a, b_1, \dots, b_n$  and  $c_1, \dots, c_m$  are ground atoms.  $a$  is called the head of the rule, denoted by  $hd(R)$ . We write  $body^+(r)$  to denote the set of positive atoms  $b_1, \dots, b_n$  and  $body^-(r)$  to denote the set of negated atoms  $c_1, \dots, c_m$ . To denote the whole body of a rule  $R$ , we use  $body(r) = body^+(r) \cup body^-(r)$ . If the body is empty, the rule is a fact. An empty head means that the rule implies  $\perp$ .

Given the definition of normal logic program rules, we can now define the *stable model semantics* for such a logic program. Due to the nature of the weak negation, it may happen that a rule is derivable at first and then other rules derive some of the negated atoms.

**Example 4.2.4.** *This can be shown in a short example:*

$$\begin{aligned} a &\leftarrow \text{not } b \\ b &\leftarrow a \end{aligned}$$

*At first glance we can deduce  $a$  from the first rule, as  $b$  is not derived so far. Next we can deduce  $b$  due to the second rule. With  $b$  deduced we would not be allowed to apply the first rule at all.*

The problem shown in the above example is the reason why it is not sufficient to compute the consequence as it has been done for definite logic programs. We need to extend the the definition for the semantics, such that the volatile nature of the negation is taken into account.

**Definition 4.2.5** (Stable Model Semantic for Normal Logic Programs). *Let  $S$  be a finite set of atoms and  $P$  a normal logic program.*

- *$S$  is closed under  $P$  if and only if  $a \in S$  whenever there is a rule  $r \in P$  such that  $head(r) = a$ ,  $body^+(r) \subseteq S$ , and  $body^-(r) \cap S = \emptyset$ .*
- *Let  $R$  be a derivation of rules of  $P$ . The set  $S$  defeats a rule  $r_i \in R$  if and only if  $S \cap body^-(r_i) \neq \emptyset$ . A valid derivation in  $S$  contains no defeated rules.*
- *$S$  is grounded in  $P$  if and only if  $a \in S$  implies that  $a \in atoms_d(R)$  holds in one derivation of  $P$  valid in  $S$ .*



$S$  is called a stable model of  $P$  if it is closed and grounded in  $P$ . Such a set is also called an answer set. We will write  $AS(P)$  to denote the set of answer sets of  $P$ .

Note that in the definition about the stable models only sets are used and that the only sequence is only to check if it is derivable. Therefore we have no direct ordering for which rule will be taken first into account if there are more than one applicable. That means different choices are possible and each choice with results in an answer set is a valid choice done.

**Example 4.2.6.** *Lets assume the following program  $P_1$ :*

$$\begin{aligned} a &\leftarrow \text{not } b. \\ b &\leftarrow \text{not } a. \end{aligned}$$

*Its two answer sets are  $AS(P_1) = \{\{a\}, \{b\}\}$ . These are the only two sets which are grounded in and closed under  $P_1$ . Intuitively we can say that it matters which rule is used first, as the “usage” of one is to inhibit the usage of the other one. In addition it is also not allowed to not use any of these rules at all, as this would be against the closedness property too.*

Another extension of the formalism would be the *disjunctive logic program*, which allows disjunctions in the head of rules. The idea behind that extension is that if the rule is derived then any number of atoms from the head, but at least one of them need to become true. Note that only minimal stable models with respect to the set of possible chosen head atoms is considered to be a stable model for disjunctive logic programs.

## 4.2.2 Algorithms for Answer Set Programming

With the definitions for an answer set program we will now go on and recapitulate some algorithms, which have been proposed during the last decades. These are either interesting as they provide an easy approach to check whether a set is an answer set or they provide a deterministic way to compute (and even enumerate) answer sets.

### Gelfond-Lifschitz-Reduct

Michael Gelfond and Vladimir Lifschitz were the first who proposed the stable model semantics for logic programs. They have introduced a usable algorithm to check whether one set is a stable model or not too. This method has later been named after the authors and is known as the *Gelfond-Lifschitz-Reduct* [Gelfond and Lifschitz, 1988].

**Definition 4.2.7** (Gelfond-Lifschitz-Reduct). *Let  $S$  be a set of atoms and  $P$  be a normal logic program. To compute the Gelfond-Lifschitz-Reduct the following steps need to be done:*

- 1)  $P' = \{r \mid r \in P \wedge \text{body}^-(r) \cap S = \emptyset\}$
- 2) Rewrite all rules  $r \in P'$ , such that the new rule  $r' = \text{head}(r) \leftarrow \text{body}^+(r)$  only consists of positive rule bodies.

The resulting definite logic program is called the *Gelfond-Lifschitz-Reduct*. We will write  $P^S$  to denote the Gelfond-Lifschitz-Reduct of  $P$  with respect to  $S$ .

The basic idea behind that approach is to eliminate all weak negations and therefore get a definite logic program, which has exactly one deduction. In order to stay valid, the reduct first removes all rules, which are invalid. To check if a set  $S$  is a stable model of a program  $P$ , the consequence of the reduct needs to be equal with the set  $S$  (i.e. if  $S = \text{Cn}(P^S)$  holds, then  $S$  is a stable model of  $P$ ).

### Smodels Algorithm

Checking whether one set is a stable model or not is a good approach, but we are more interested in computing one or even enumerating all stable models of a given program. There are other algorithms and theories available (e.g. Faber-Leone-Pfeifer (FLP) reduct [Faber et al., 2004]), though we will present one easy to understand approach: the Smodels Algorithm [Niemelä and Simons, 1996]. The basic idea is to use monotonic reasoning as long as possible. Then one nondeterministic choice is done, followed by as much monotonic reasoning as possible. To represent that computational flow, the whole algorithm is represented as a binary tree. Each monotonic reasoning step will add exclusively one child node to a given node in the tree, while the nondeterministic choice is represented by two child nodes. They also use the antichain property of answer sets, which means that an answer set cannot be a subset of another answer set.

**Proposition 4.2.8.** *Let  $P$  be a normal logic program and the sets  $S$  and  $S'$  are answer sets of  $P$ . Then  $S \subseteq S'$  implies that  $S = S'$ .*

*Proof.* Assume that  $S \subseteq S'$  hold. As both sets are answer sets of  $P$ , it holds that  $S = \text{Cn}(P^S)$  and that  $S' = \text{Cn}(P^{S'})$ .  $S \subseteq S'$  implies that  $P^S \supseteq P^{S'}$ , because  $S$  has potentially fewer rules which will get removed by the first step of the Gelfond-Lifshitz-Reduct than those potentially removed by the bigger set  $S'$ . Therefore, due to the fact that the consequences of definite logic programs are monotone, it also holds that  $\text{Cn}(P^S) \supseteq \text{Cn}(P^{S'})$ , what means that  $S \supseteq S'$  holds too. From the first assumption  $S \subseteq S'$  and the last conclusion  $S \supseteq S'$  we can follow, that if  $S \subseteq S'$  hold, then  $S = S'$  must hold too.  $\square$

Another idea of the algorithm is to find the biggest subset of atoms, which has to be part of an answer set, while identifying those atoms which cannot be part of an answer set. This idea is applied by the use of (*LU*)-answer sets.

**Definition 4.2.9** (( $LU$ )-Answer Set). *Let  $P$  be a normal logic program and  $L \subseteq U$  be sets of atoms.  $S$  is a ( $LU$ )-answer set of  $P$  if  $S$  is an answer set of  $P$  and  $L \subseteq S \subseteq U$  hold.*

By using  $Atoms(P)$  to denote all atoms occurring in a program  $P$ , it is easy to see that every answer set of  $P$  is a ( $LU$ )-answer set of  $(\emptyset, Atoms(P))$ . Note that the notion of  $L$  and  $U$  is chosen to denote the lower and upper bound for answer sets which are in between these two boundaries. First we will provide the *expand algorithm*, which is the monotonic reasoning part of the Smodels algorithm.

**Definition 4.2.10** (Expand).

*Expand*( $LU$ )

*repeat*

$L' := L$

$U' := U$

$L := L \cup Cn(P^{U'})$

$U := U \cap Cn(P^L)$

*until*  $L = L'$  **and**  $U = U'$

Basically, elements in  $U$  are candidate atoms for elements of an answer set and atoms in  $L$  are already necessary atoms to be in an answer set. All atoms which are neither in  $U$  nor in  $L$  are those which are already refuted and shall not be in an answer set. The overall algorithm works as follows.

**Definition 4.2.11** (Smodels Algorithm).

- 1) Create a binary tree with the root node  $(\emptyset, Atoms(P))$
- 2) Expand each leaf node  $(L, U)$  to get an expanded child node  $(L', U')$
- 3) Choose for each expanded leaf node an atom  $a \in (U' \setminus L')$  and branch the tree with nodes  $(L' \cup \{a\}, U')$  and  $(L', U' \setminus \{a\})$
- 4) If  $L' = U'$  holds after point 2), then an answer set is found
- 5) if  $U' \subset L'$  then an inconsistency of the branch occurred and the branch cannot result in a valid answer set

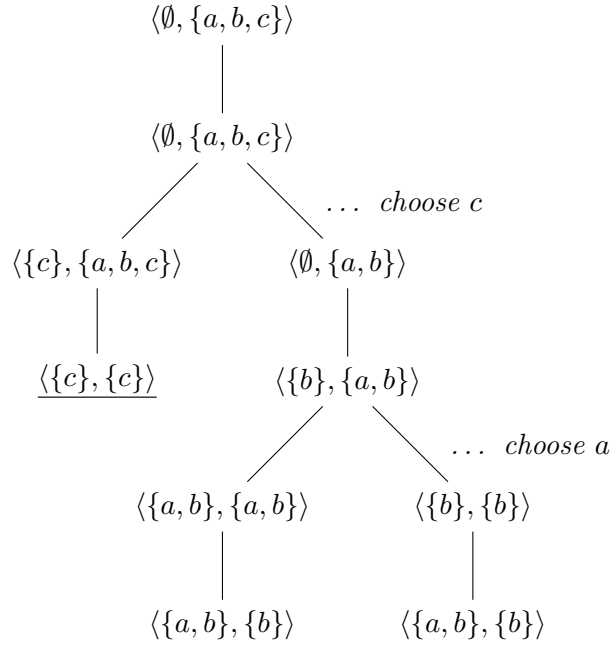
**Example 4.2.12.** *Lets assume the following normal logic program  $P$ :*

$a \leftarrow b, \text{not } a$

$b \leftarrow \text{not } c$

$c \leftarrow \text{not } b$

*We will compute the stable models with the help of the Smodels algorithm, as presented above:*



*Expand*  $\langle \emptyset, \{a, b, c\} \rangle$ :

$$L = \emptyset \cup \emptyset = \emptyset$$

$$U = \{a, b, c\} \cap \{a, b, c\} = \{a, b, c\}$$

*Expand*  $\langle \{c\}, \{a, b, c\} \rangle$ :

$$L = \{c\} \cup \emptyset = \{c\}$$

$$U = \{a, b, c\} \cap \{c\} = \{c\}$$

$$L = \{c\} \cup \{c\} = \{c\}$$

$$U = \{c\} \cap \{c\} = \{c\}$$

*Expand*  $\langle \emptyset, \{a, b\} \rangle$ :

$$L = \emptyset \cup \{b\} = \{b\}$$

$$U = \{a, b\} \cap \{a, b\} = \{a, b\}$$

$$L = \{b\} \cup \{b\} = \{b\}$$

$$U = \{a, b\} \cap \{a, b\} = \{a, b\}$$

*Expand*  $\langle \{a, b\}, \{a, b\} \rangle$ :

$$L = \{a, b\} \cup \{b\} = \{a, b\}$$

$$U = \{a, b\} \cap \{b\} = \{b\}$$

$$L = \{a, b\} \cup \{a, b\} = \{a, b\}$$

$$U = \{b\} \cap \{b\} = \{b\}$$

*Expand*  $\langle \{b\}, \{b\} \rangle$ :

$$L = \{b\} \cup \{a, b\} = \{a, b\}$$

$$U = \{b\} \cap \{b\} = \{b\}$$

$$L = \{a, b\} \cup \{a, b\} = \{a, b\}$$

$$U = \{b\} \cap \{b\} = \{b\}$$

The algorithm produces the only answer set for  $P$ , which is  $AS(P) = \{\{c\}\}$ .

### 4.2.3 Answer Set Programming in Practice

We have introduced answer set programming on ground programs so far. For better usability, the language of answer set programming has been standardised

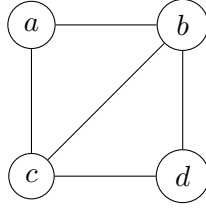
in the *ASP-Core-2-Standard* [Calimeri et al., 2015]. We will now have a look at the most important additions with respect to the already presented formal semantics.

In a real application someone would be interested in programs which do feature the use of variables. As there are no algorithms to derive the answer set directly from a program with variables, it is always imperative to ground the program first. This grounding process is generally just the substitution of each variable with all of its ground instances. To limit the domain for each variable and to have better readability it is usual to use predicates (i.e. as in first order logic). Another convention is to have the first letter of the variable name in upper case, while predicates and grounded atoms start with a lower letter. Most systems have the grounder as a feature of their solver, while there are others, like GRINGO [Gebser et al., 2011c] which is only responsible for grounding. In addition to variables, most solvers support the use of basic arithmetic functionality like basic arithmetic operators and comparators. Another addition is the use of aggregates and pools of predicates, from which some might be chosen or filtered out.

Note that different implementations of the same program may have huge differences in the size of the grounded program. Therefore there are some modelling techniques and design patterns which try to reduce the size of the grounded program. In general it is a good idea to keep the arity of predicates low to minimise cross-products instantiations of domains. It is also advised to check inequalities for symmetric problems by using the  $<$  relation instead of  $\neq$ . This is called diagonalisation and lets well written grounder remove about half of the rules generated by the grounding process, as these cannot occur at all (e.g.  $5 < 4$  may never happen, while  $5 \neq 4$  is still a valid possibility).

Most problems to solve with answer set programming are at least **NP**-hard. To solve these problems the concept of *guess & check* is used. First all possible (and roughly reasonable) solutions are guessed, where each answer set represents one of these candidates. Then all wrong guesses, are removed by marking them as inconsistent answer sets, such that they are no longer stable model candidates. This is done by using *constraints*. Such a constraint is a rule, with an empty head. If the rule may be derived, the whole answer set candidate is no longer considered as valid and will not be part of the result of the program.

**Example 4.2.13.** *To show how such an answer set program for an NP-hard problem might look like, we will now try to solve the 3-colouring of an arbitrarily given undirected graph. Each vertex will be represented by the unary predicate  $v$ , and each edge by the binary predicate  $e$ . So one example-graph might look as follows:*



The representation for the answer set program would be:

$$\text{graph} = \{v(a)., v(b)., v(c)., v(d)., e(a, b)., e(a, c)., e(b, c)., e(b, d)., e(c, d).\}$$

To get an answer set program which solves the 3-colouring problem, we will now guess a colouring and afterwards we will check if the condition for a valid 3-colouring, that is that two adjacent vertices are not allowed to be in the same colour, hold.

$$\begin{aligned} 3 - \text{col-guess} &= \{ \text{col}(V, r) \leftarrow v(V), \text{not col}(V, b), \text{not col}(V, g), \\ &\quad \text{col}(V, b) \leftarrow v(V), \text{not col}(V, r), \text{not col}(V, g), \\ &\quad \text{col}(V, g) \leftarrow v(V), \text{not col}(V, r), \text{not col}(V, b) \} \\ 3 - \text{col-check} &= \{ \perp \leftarrow e(A, B), \text{col}(A, X), \text{col}(B, X) \} \end{aligned}$$

$AS(\text{graph} \cup 3 - \text{col-guess} \cup 3 - \text{col-check})$  is the set of all possible 3-colourings for the graph. Note that if there is no answer set, then no 3-colouring exists.

There are also further advanced techniques which may be used for answer set programming. By utilising disjunctive logic programs one might also solve problems in  $\Sigma_2^P$ . A special technique, called *saturation* [Eiter and Gottlob, 1995] is especially handy for this kind of problems. There a nested **NP**-Problem might be solved with the help of a disjunctive guess. To describe the idea more intuitively we will use the tautology problem for a formula as an example for the nested problem. At first each variable is guessed via a disjunction to generate all possible interpretations. Then each interpretation is checked if it is satisfiable. Note that no weak negation might occur during these checks. If it is satisfiable a new atom “satisfied” is introduced via a rule. Afterwards it is checked with weak negation if “satisfied” could be deduced. In case it could not be deduced, the answer set is set unsatisfiable (via a constraint). If it is satisfiable, the answer set will be saturated, which means that all atoms from the disjunctive guess are set to true. This works because the “satisfied” atom is derived only by monotonic rules (i.e. no weak negation) and the addition of more atoms does not change its outcome. In addition all “satisfied” alternatives will collapse to one answer set. If all interpretations are satisfied, only one answer set remains, the one where “satisfied” is true and all atoms are considered true. If at least one interpretation is not satisfied, then this one is not saturated and therefore a smaller (with respect to subset inclusion) answer set for the disjunctive program. In that case this one, smaller (with respect

to subset inclusion) would be the one answer set which should be chosen. As this is due to the constraint inconsistent, there is no interpretation to go on. That is the way to check if a nested tautology problem still holds. Note that this technique may lead to exponential memory consumption of the solver, as it has to check every (potentially exponential) interpretation.

Completing the part about logic programming, we will now have a look at actually usable solvers. We will have a look at the two most commonly used solvers in the scientific community. Both of them are implementing the *Core-ASP-2*-standard as a basis and add different additional features on top of it. CLASP, which is part of the *Potsdam Answer Set Solving Collection* (POTASSCO) [Gebser et al., 2011b] utilises a similar but way more specialised approach than the *Smodels* algorithm. The collection has a plethora of different tools, such as GRINGO for grounding, CLASP for solving, CLINGO as a combination of the former two, different optimisation tools and much more. The other system is to compute Answer Set Programs with external sources. It is called DLV-HEX [Redl, 2014] and is an extension to classical answer set programming. This paradigm allows it to use predicates which are externally computed during the computation of the answer sets. In addition it might use already derived information from the answer set solver. While POTASSCO also has some interaction with external sources in the form of *online-clingo* (OCLINGO), they are focused on considering streamed information which are taken into account during a sequence of succeeding computations of answer sets.

We will mostly use CLINGO in following applications, but DLV-HEX is an interesting approach which could be utilised very easily because its infrastructure changed and allows CLINGO to be used as the internal solving engine.

### 4.3 Abstract Dialectical Frameworks

*Abstract Dialectical Frameworks* (ADF) [Brewka and Woltran, 2010] are a natural generalisation of Dung’s Argumentation Frameworks. In Dung’s Frameworks it is only possible to declare a conflict between two arguments as an attack. One might ask, why another and more complicated formalism is of interest if there is already a well established and accepted framework in the form of Dung’s Argumentation Frameworks. We are aware of at least two reasons, why a generalisation might be preferable:

- the generalisation is more expressive,
- the generalisation allows easier modelling of problems.

It is a fact, that both of these reasons are applicable for abstract dialectical frameworks. We will have a look into both matters later on in this section (cf. 4.3.2).

The basic idea of *ADF*s is to extend the possible relations between arguments, such that any kind might be possible. That means it is possible to define attack, support, and conditional behaviour. To give an easier to grasp picture on that idea, let's suppose that we have some arguments  $a, b, c$ , and  $d$ . Further we want to say that  $a$  supports  $d$ ,  $b$  attacks the standpoint of  $d$ , and  $c$  will only support  $d$  if  $b$  is refuted. Figure 4.3 shows one possible visual

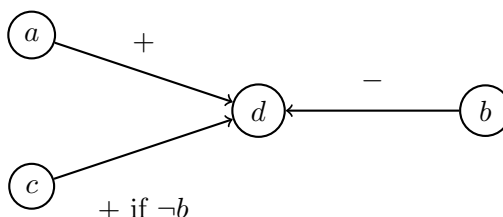


Figure 4.2: Visual representation of the idea of an *ADF*

representation of that idea and presented example of such an *ADF*. We can now represent the acceptance of  $d$  as a propositional formula, where the atoms correspond to the arguments of the framework. In our example, the formula for  $d$  could be  $c \iff \neg b$ . One can already see by the example that intricate positions between different arguments can be defined and modelled in an easy way. In addition it is possible to have direct support or conditional standpoints to another argument, which allows more freedom to relate arguments to each other.

The first work got further analysed by [Ellmauthaler, 2012] and based on some of these insights it got revisited and was formally overhauled in a followup paper [Brewka et al., 2013]. We will only focus on the revisited variant of the framework, as it is described in the newly published handbook chapter [Brewka et al., 2017b]<sup>12</sup>. Building upon the algebraic framework for studying semantics of knowledge representation formalisms, introduced by Denecker, Marek, and Truszczyński [Denecker et al., 2000] the new semantics for *ADF*s have been defined. This has been discussed, investigated, and defined by Strass in [Strass, 2013]. In the study of semantics for knowledge representation formalisms objects of interest may be represented by elements of lattices. The important and significant work of Denecker, Marek, and Truszczyński was to approximate an operator. Such operators will transform objects of interest into others, based on the given knowledge base they are working on. A fixpoint of such an operator is the point where the object is not transformed further by additional applications of it. Informally speaking that means that the operator can neither add or retract information from the object, based on the information taken from the knowledge base. The interesting point on this approximation of fixpoints

<sup>12</sup>The chapter has been published as a Journal Paper because the book is to appear.



is, that if the operator is operating on two-valued interpretation, then the approximation will be defined upon a three-valued interpretation.

In the following, we will present the definitions for *ADF*s, followed by an overview on computational aspects of *ADF*s. Afterwards we will discuss existing systems and will have a detailed look at one of their systems. For more detailed insights and proofs we refer the interested reader to the Journal-version of the Argumentation-Handbook [Brewka et al., 2017b].

### 4.3.1 Syntax and Semantics

As already seen in the example above, an *ADF* can be seen as a directed graph. Dung's Argumentation Frameworks are a standard where this is done in a similar way. We see vertices as a representation of arguments, statements, or positions. Such a statement is an arbitrary item, which might be accepted or rejected. The directed edges are representing some kind of dependency between two statements. Acceptability of a statement  $s$  only depends on the status of its parents (denoted by  $par(s)$ ), which is done by an edge from each parent to the statement in question. Unlike attack relations in Dung Argumentation Frameworks, in *ADF*s the meaning of an edge may vary. To specify the exact conditions for the statement  $s$  to be accepted, an *acceptance condition*  $C_s$  is associated with each statement. This acceptance condition is a function, which maps to each subset of  $par(s)$  one of the truth values  $\mathbf{t}$  or  $\mathbf{f}$ <sup>13</sup>.

**Definition 4.3.1** (Abstract Dialectical Framework). *An Abstract Dialectical Framework is a tuple  $D = (S, L, C)$  where*

- $S$  is a set of statements,
- $L \subseteq S \times S$  is a set of links, and
- $C = \{C_s\}_{s \in S}$  is a set of total functions  $C_s : 2^{par(s)} \mapsto \{\mathbf{t}, \mathbf{f}\}$ , one for each statement  $s$ . We call  $C_s$  the acceptance condition of  $s$ .

It is in many cases more convenient to use a different representation of the acceptance condition. Therefore we will use a propositional formula to define the function instead. So the logical representation of an *ADF*  $(S, L, C)$  is as in definition 4.3.1, but  $C$  is a collection  $\{\varphi_s\}_{s \in S}$  of propositional formulae. Each acceptance condition  $C_s$  is then a propositional formula  $\varphi_s$  with  $par(s)$  as its domain. Note that as the acceptance condition defines how the relation between linked statements look like, it is in general redundant information. If not stated otherwise, we will omit  $L$  and say that all statements in  $\varphi_s$  for a statement  $s \in S$  are the set of all  $par(s)$ .

<sup>13</sup>First the values **In** and **Out** has been used. For easier application of formal logic and better readability, this has been changed

**Example 4.3.2.** *During this section we will use the following ADF, which will act as some small and easy to grasp running example. We use the graphical representation in Figure 4.3 of the ADF for better readability. Intuitively  $\varphi_a$  can*

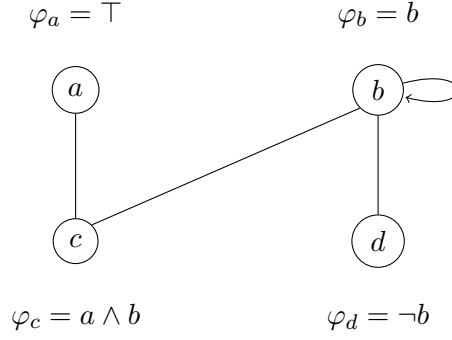


Figure 4.3: Example ADF

be read that  $a$  should always be acceptable.  $\varphi_b$  states some sort of self-support, which can be seen as a guess whether it should be accepted or not. Another point of view might be that this statement has no external reason to be accepted and it depends on the interpretation whether self-support is sufficient or not.  $\varphi_c$  is representing that  $c$  should be accepted if  $a$  as well as  $b$  is accepted. Finally,  $d$  is only accepted if  $b$  is not accepted i.e.  $d$  is attacked by  $b$ .

To define the various semantics (i.e. the ADF versions of those presented in section 3.3 for Dung’s Argumentation Frameworks) of ADFs over statements  $S$  we use the approximation fixpoint theory on the notion of a *two-valued* model. The total function  $v : S \mapsto \{\mathbf{t}, \mathbf{f}\}$  is a two-valued interpretation, which maps to each statement either one of the truth values  $\mathbf{t}$  or  $\mathbf{f}$ . We call such an interpretation a model, if for every statement  $s \in S$  it holds that  $v(s) = v(\varphi_s)$ <sup>14</sup>. In other words, a model is an interpretation which maps truth values to the statements such that their acceptance condition evaluates to the same result as it is for the value of the associated statement.

### Semantics of ADFs via Approximation Fixpoint Theory

We will now utilise the Approximation Fixpoint Theory and will show how this is done for ADFs. The theory is dealing with operator-based semantics and provides a way to approximate them. We will start with a two-valued operator,

---

<sup>14</sup>Note that  $v(\varphi_s)$  is the valuation of that propositional formula under the given interpretation

which takes a two-valued interpretation and returns an updated two-valued interpretation.

**Definition 4.3.3** (Operator  $G_D$ ). *Let  $D = (S, \{\varphi_s\}_{s \in S})$  be an ADF. The operator  $G_D : V_2 \mapsto V_2$  maps one interpretation  $v : S \mapsto \{\mathbf{t}, \mathbf{f}\}$  to an updated interpretation, such that*

$$G_D(v) : S \mapsto \{\mathbf{t}, \mathbf{f}\} \text{ where } s \mapsto v(\varphi_s)$$

That operator takes some given interpretation, evaluates every acceptance condition according to that interpretation, and returns an interpretation which matches the results of the evaluations. This is in fact a characterisation of the two valued model semantics we have defined above.

**Proposition 4.3.4.** *Let  $D = (S, \{\varphi_s\}_{s \in S})$  be an ADF and  $v : S \mapsto \{\mathbf{t}, \mathbf{f}\}$  be a two valued interpretation. Then  $v$  is a two valued model of  $D$  if and only if  $v = G_D(v)$ .<sup>15</sup>*

To apply the definitions for an ultimate approximation in their approximation fixpoint theory from Denecker, Marek, and Truszczyński [Denecker et al., 2004], we need to introduce the third value for three valued operators and interpretations first. With two truth values, we have that something is either true ( $\mathbf{t}$ ) or false ( $\mathbf{f}$ ). During computation or even on some occasions it is (still) not decided which of these two values should be assigned. That is why we use the value *undecided* ( $\mathbf{u}$ ) as our third possible valuation. It is easy to see that while true and respectively false carry much information with each statement, there is less information in the statement that something is undecided.

We will reflect that difference in information by introducing an information ordering  $\leq_i$ . This ordering will assign a direct information order  $\{\mathbf{t}, \mathbf{f}\}$  with respect to  $\mathbf{u}$ , such that  $\mathbf{u} <_i \mathbf{t}$  and  $\mathbf{u} <_i \mathbf{f}$ .  $\leq_i$  is the symmetric transitive closure of  $<_i$ . Note that  $\mathbf{t}$  and  $\mathbf{f}$  are not comparable in  $\leq_i$ , so neither  $\mathbf{f} \leq_i \mathbf{t}$  nor  $\mathbf{t} \leq_i \mathbf{f}$  hold. The partially ordered set  $(\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}, \leq_i)$  forms a complete meet-semilattice with the meet operator  $\sqcap_i$ .<sup>16</sup> The meet operator evaluates two values such that  $\mathbf{t} \sqcap_i \mathbf{t} = \mathbf{t}$ ,  $\mathbf{f} \sqcap_i \mathbf{f} = \mathbf{f}$ , and every thing else evaluates to  $\mathbf{u}$ . It can be seen as some kind of *consensus* operator, which needs both arguments to agree on some truth value. Additionally we can now extend the partial order to interpretations as well, such that

$$v_1 \leq_i v_2 \text{ if and only if } \forall s \in S : v_1(s) \in \{\mathbf{t}, \mathbf{f}\} \Rightarrow v_1(s) = v_2(s).$$

Therefore we can also extend the  $\sqcap_i$ -operator to interpretations:

$$\forall s \in S : (v_1 \sqcap_i v_2)(s) = v_1(s) \sqcap_i v_2(s)$$

<sup>15</sup>For proof see [Strass, 2013]

<sup>16</sup>A complete meet-semilattice is such that every non-empty finite subset has a greatest lower bound, the meet; and every non-empty directed subset has a least upper bound. A subset is directed iff any two of its elements have an upper bound in the set.

We will write  $[v]_2$  to describe the set of all two valued valuations which have more or equal information with respect to  $\leq_i$  to a given three valued interpretation  $v$ .

**Corollary 4.3.5.** *Let  $D$  be an ADF. The operator  $\Gamma_D : V_3 \mapsto V_3$  is the ultimate approximation of  $G_D$  and assigns thus: for an ADF  $D$  and a three valued interpretation  $v$ , the revised interpretation  $\Gamma_D(v)$  is given by*

$$\Gamma_D(v) : S \mapsto \{\mathbf{t}, \mathbf{f}, \mathbf{u}\} \text{ with } s \mapsto \bigcap_i \{w(\varphi_s) \mid w \in [v]_2\}.$$
<sup>17</sup>

Based on this result we can now define the semantics for ADFs. Note that there also exists an approximation fixpoint theory operator for Dung's Argumentation Frameworks, so we will use the dual definitions just with the operator  $\Gamma_D$  for ADFs.

**Definition 4.3.6** (Semantics for ADFs). *Let  $D = (S, \{\varphi_s\}_{s \in S})$  be an ADF and  $v : S \mapsto \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$  be an interpretation.*

- (i)  $v$  is admissible for  $D$  if and only if  $v \leq_i \Gamma_D(v)$
- (ii)  $v$  is complete for  $D$  if and only if  $v = \Gamma_D(v)$
- (iii)  $v$  is preferred for  $D$  if and only if  $v$  is  $\leq_i$ -maximal admissible
- (iv)  $v$  is grounded for  $D$  if and only if  $v$  is the  $\leq_i$ -least fixpoint of  $\Gamma_D$

The operator  $\Gamma_D$  seems a little bit unintuitive at first glance. For the sake of easier understanding it can be described with another approach. Given a three-valued interpretation  $v$ , we will substitute in every  $\varphi_s$  for  $s \in S$  of an ADF the variables which are mapped to  $\{\mathbf{t}, \mathbf{f}\}$  with  $\top$  and  $\perp$  respectively to get  $\varphi_s^v$ . Then  $\Gamma_D$  can be defined as follows:

$$\Gamma_D(v) : S \mapsto \{\mathbf{t}, \mathbf{f}, \mathbf{u}\} \text{ with } s \mapsto \begin{cases} \mathbf{t} & \text{if } \varphi_s^v \text{ is a tautology} \\ \mathbf{f} & \text{if } \varphi_s^v \text{ is unsatisfiable} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

To represent such three valued interpretations more easily, we will use the following, more convenient notation. Such an interpretation is represented as a set of literals, such that

- every statement which is mapped to  $\mathbf{t}$  occurs as a positive literal,
- every statement which is mapped to  $\mathbf{f}$  occurs as a negated literal, and
- every statement which is mapped to  $\mathbf{u}$  will be omitted from the set.

---

<sup>17</sup>For proof see [Strass, 2013]

**Example 4.3.7.** Assume one possible interpretation for the ADF from Example 4.3.2 which might be

$$v_1 = \{a \mapsto \mathbf{t}, b \mapsto \mathbf{t}, c \mapsto \mathbf{u}, d \mapsto \mathbf{f}\}$$

The shortened convenience notation for that interpretation is

$$v_1 = \{a, b, \neg d\}$$

To define the *stable model* semantics for ADFs [Brewka et al., 2013], we have borrowed the idea from the Gelfond-Lifschitz-Reduct for Logic Programming. We do need some similar approach as ADFs are not only utilising attacks, but also support type relations. The reduct approach is utilised to exclude self-justifying cycles. Similarly to logic programming, we start by guessing some two valued model for the Framework. Then we eliminate all statements which are evaluated to false by the model. Next we replace all occurrences of removed statements in the acceptance conditions by  $\perp$ . Depending on the representation of the ADF, one also needs to revise the links as well. Finally we check if all statements mapped to  $\mathbf{t}$  by the two valued model coincide with the result of the grounded interpretation of the reduct. As the grounded interpretation is the least fixpoint, there is no possibility for any statement valuated to  $\mathbf{t}$  might be interpreted in another valid way.

**Definition 4.3.8** (Reduced ADF). Let  $D = (S, L, C)$  be an ADF with  $C = \{\varphi_s\}_{s \in S}$  and  $v : S \mapsto \{\mathbf{t}, \mathbf{f}\}$  be a two-valued model of  $D$ . Define the reduced ADF  $D^v$  with  $D^v = (S^v, L^v, C^v)$ , where

- $S^v = \{s \mid s \in S \wedge v(s) = \mathbf{t}\}$
- $L^v = L \cap (S^v \times S^v)$
- $C^v = \{\varphi_s^v\}_{s \in S^v}$  where for each  $s \in S^v$ , we set  $\varphi_s^v = \varphi_s[b/\perp : v(b) = \mathbf{f}]$ .

We denote  $w$  as the unique grounded interpretation of  $D^v$ . A two-valued model  $v$  is a stable model of  $D$  if and only if for all  $s \in S$ , it holds that  $v(s) = \mathbf{t} \Rightarrow w(s) = \mathbf{t}$ .

**Example 4.3.9.** Consider the ADF  $D$ , given by the acceptance conditions

$$\varphi_a = \top, \varphi_b = \neg a \vee c, \varphi_c = b.$$

This ADF has two models, which are  $v_1 = \{a, b, c\}$  and  $v_2 = \{a, \neg b, \neg c\}$ . By checking whether they are stable, we need to get the reduction  $D^{v_1}$ , which is identical to  $D$ . The grounded interpretation of  $D$  is  $\{a\}$ , which implies that  $v_1$  is not stable. The other model produces the reduction  $D^{v_2} = (S^{v_2}, L^{v_2}, C^{v_2})$  with  $S^{v_2} = \{a\}$ ,  $L^{v_2} = \emptyset$ , and  $\varphi_a = \top$ . The grounded interpretation of  $D^{v_2}$  is  $\{a\}$ , which implies that this is a stable model. Note that the first model is rightfully no stable model, as the statements  $b$  and  $c$  are in a self justifying circle, such that  $b$  is only acceptable because of  $c$  and  $c$  is only because of  $b$  acceptable.

### Relation between Semantics

*ADF*s are a generalisation of Dung Argumentation Frameworks, therefore the same relation between the semantics should hold for *ADF*s. First we will have a look on the *ADF* from Example 4.3.2 and then we will present the formalisation of the Theorem 4.3.11, as done in [Brewka et al., 2013].

**Example 4.3.10.** For the *ADF* from Example 4.3.2 we are going to present the different interpretations in an easy to view manner in Figure 4.3.10. The figure

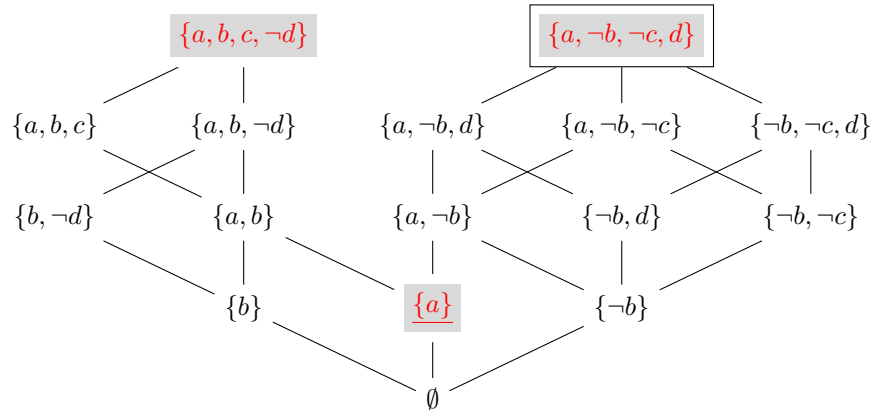


Figure 4.4: admissible, ground, **complete**, and preferred interpretation, as well as the stable models for the *ADF* from Example 4.3.2

shows all admissible interpretations of the *ADF*, which are ordered after the number of  $\mathbf{u}$  mappings. Every vertical level has the same number of statements valuated to  $\{\mathbf{t}, \mathbf{f}\}$ . In addition if there is a line between two interpretations, that means that  $\leq_i$  holds between these two. Based on this visualisation of admissible interpretations we have marked the unique grounded interpretation by underlining it. Complete interpretations are coloured in red and have a darker background. The preferred interpretations are all interpretations on the top level. As the top level contains no  $\mathbf{u}$  mappings in our example, that also means that these interpretations are two-valued models. To visualise which of these models are stable models, we have added a box around the stable interpretation.

**Theorem 4.3.11** (from [Brewka et al., 2013]). Let  $D$  be an *ADF*.

- Each stable model of  $D$  is a two-valued model of  $D$ ;
- Each two-valued model of  $D$  is a preferred interpretation of  $D$ ;
- Each preferred interpretation of  $D$  is complete;
- Each complete interpretation of  $D$  is admissible;

- *The grounded interpretation of  $D$  is complete.*

Note that a Dung Argumentation Framework can be easily transformed into an *ADF*, by constructing an associated *ADF*.

**Definition 4.3.12** (Associated *ADF*). *For a Dung Argumentation Framework  $F = (A, R)$ , the associated *ADF*  $D_F = (A, R, C)$  with  $C = \{\varphi_a\}_{a \in A}$  such that for each  $a \in A$ , the acceptance condition is given by*

$$\varphi_a = \bigwedge_{\substack{b \in A \\ (b,a) \in R}} \neg b$$

### 4.3.2 Computational Aspects

We have discussed how *ADFs* allow different kind of relations between statements. On an informal level we used the notion and concept of attack, support and dependency. Indeed we are interested in a formal definition, which has been done already in the first paper [Brewka and Woltran, 2010]. Later the additional notion of redundant and dependent links has been analysed further [Ellmauthaler, 2012].

**Definition 4.3.13** (Link Types). *Let  $D = (S, L, C)$  be an *ADF*. A link  $(r, s) \in L$  is*

- *supporting in  $D$  if and only if for all  $R \subseteq \text{par}(s)$ , we have  $C_s(R) = \mathbf{t}$  implies  $C_s(R \cup \{r\}) = \mathbf{t}$ ,*
- *attacking in  $D$  if and only if for all  $R \subseteq \text{par}(s)$ , we have  $C_s(R \cup \{r\}) = \mathbf{t}$  implies  $C_s(R) = \mathbf{t}$ ,*
- *redundant in  $D$  if and only if it is attacking and supporting in  $D$ , and*
- *dependent in  $D$  if and only if it is neither attacking nor supporting in  $D$ .*

$L^+ \subseteq L$  denotes the set of all supporting statements, and  $L^- \subseteq L$  is the set of all attacking links in  $L$ .

An *ADF* is bipolar (a *BADF*) if all links in  $L$  are supporting or attacking (or both) (i.e.  $L = L^+ \cup L^-$ ). In other words in *BADFs* no dependent links are allowed. The interesting point about this class of *ADFs* is how it positions it from the point expressiveness and computational complexity in between general *ADFs* and Dung Argumentation Frameworks.

First let us formalise what we understand by *expressiveness*. Given a formalism  $\mathcal{F}$  we are interested in the set of structures which can be defined with semantics  $\sigma$  over a vocabulary  $A$ . This is called the signature of a formalism  $\mathcal{F}$  with respect to semantics  $\sigma$ , which is

$$\Sigma_{\mathcal{F}}^{\sigma} = \{\sigma(kb) \mid kb \in \mathcal{F}\}.$$

Intuitively it is a characterisation of what can (or cannot) be done with a formalism  $\mathcal{F}$  under semantics  $\sigma$  [Gogic et al., 1995]. To analyse the relation between two formalism  $\mathcal{F}_1$  and  $\mathcal{F}_2$  which share the same semantics  $\sigma$ , we can compare their sets to each other. If  $\Sigma_{\mathcal{F}_1}^\sigma \subsetneq \Sigma_{\mathcal{F}_2}^\sigma$  holds, that mean that  $\mathcal{F}_2$  is more expressive than  $\mathcal{F}_1$  (i.e. can represent everything that  $\mathcal{F}_1$  can, and more).

Dung’s Argumentation Frameworks, *BADFs*, and *ADFs* have been investigated and compared in various work [Strass, 2015b, Strass, 2015a].

**Theorem 4.3.14.** *For  $\sigma \in \{adm, com, prf, mod\}$ , we find that*

$$\Sigma_\sigma^{AF} \subsetneq \Sigma_\sigma^{BADF} \subsetneq \Sigma_\sigma^{ADF}.$$

*For the stable model semantics  $stm$ , we find that*

$$\Sigma_{mod}^{AF} = \Sigma_{stm}^{AF} \subsetneq \Sigma_{stm}^{BADF} = \Sigma_{stm}^{ADF}.$$

*Furthermore, for the model semantics we have*

$$\Sigma_{mod}^{ADF} = \mathcal{V}_2 = \{v : A \mapsto \{\mathbf{t}, \mathbf{f}\}\},$$

*that is, ADFs under the model semantics are universally expressive.*

Note that the model and stable model semantics is for Dung’s Argumentation Framework the same, as there are no support cycles which need to be dealt with. The results show that *BADFs* are almost universally more expressive than Dung Argumentation Frameworks, and that in most instances they are less expressive than *ADFs*.

Computational Complexity wise, there has been various work on analysing the complexity of the Frameworks [Ellmauthaler, 2012, Brewka et al., 2013, Strass and Wallner, 2015, Gaggl et al., 2015, Polberg and Wallner, 2017, Wallner, 2014]. As computational problems, one is interested in different kinds of reasoning tasks, such as:

- **Credulous acceptance** of a statement: is there at least one interpretation under semantics  $\sigma$ , where statement  $a$  is assigned to be true?
- **Sceptical acceptance** of a statement: is it true in every interpretation under semantics  $\sigma$ , that statement  $a$  is assigned to be true?
- **Interpretation verification**: is a given interpretation an interpretation under semantics  $\sigma$ ?
- **Existence** of an Interpretation: is there an interpretation under semantics  $\sigma$ ?

In most cases *ADFs* are shown to be one level higher on the polynomial hierarchy than Dung’s Argumentation Frameworks. For *BADFs* the complexities are in the worst case the same as those for Dung’s Argumentation



Frameworks, if the link-types are known (i.e. which links are attacking and which are supporting). Note that computing the link-type of a link is itself **coNP**-hard. Still, that complexity behaviour can be exploited if the link type can be easily deduced during the instantiation of the Framework. In general these results show that *ADF*s, and in specific cases *BADF*s are an expressive modelling tool, despite its comparable pretty high computational complexity (in comparison to Dung Frameworks).

### 4.3.3 Systems for Abstract Dialectical Frameworks

There have been developed several systems to compute interpretations for *ADF*s. Here we also want to mention further development of *ADF*s into a more graphical representation of acceptance conditions. GRAPPA [Brewka and Woltran, 2014] uses a meta language to describe how acceptance conditions might be deduced by labels given to each link. Shortly said this language defines which kind of operations are allowed and how the preferences between different links are computed. It is possible to transform a GRAPPA-instance into an *ADF*.

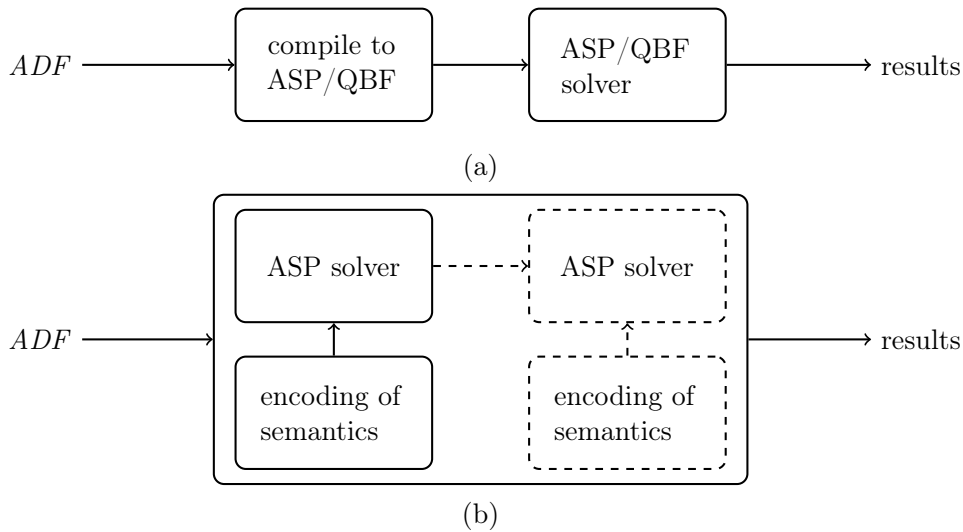


Figure 4.5: Approaches of the Workflow for different systems. (a) visualises the instance based compilation approach and (b) is showing the static encoding approach.

There are two different underlying formalisms, utilised by the different systems. The first is declarative programming in the form of Answer Set Programming and the latter is qualified Boolean formula satisfiability (QBF). For the Answer Set Programming based language there have been two different

workflows used. One is the *instance-based compilation* approach, where each a new encoding is generated for every problem. The other is the *static encoding* approach, which provides a set of static encodings which are combined with the answer set programming representation of the *ADF* and compute on that basis the stable models. Note that in both approaches the encoding is designed in such a way that the stable model corresponds to the chosen *ADF* semantics to compute. Due to the high computational complexity of *ADFs* there are systems where for some semantics more than one solver call is needed, to get a matching computational complexity of the solver. We also want to mention that some solvers apply normal logic programs, while others utilise disjunctive logic programs. The QBF approach also uses the instance based compilation approach, utilising a QBF solver instead of an Answer Set Programming solver. A graphical overview for better understanding these two logic programming approaches can be seen in Figure 4.5

QADF<sup>18</sup> [Diller et al., 2015] is a system which uses QBF formulae to compute the semantics of an *ADF* utilising the instance-based compilation approach. Based on the given *ADF* and the chosen semantics an encoding is generated, where the interpretations to the QBF formula are corresponding to the interpretations of the semantics.

The system GRAPPAVIS<sup>19</sup> [Heißenberger, 2016] is an implementation of the shortly mentioned GRAPPA framework. It is a hybrid between the instance-based compilation and the static encoding approach. First a given GRAPPA input is compiled into a declarative Answer Set Programming representation and then a static encoding computes the semantics.

Based on the static encoding of GRAPPAVIS, the system YADF<sup>20</sup> [Brewka et al., 2017a] the system compiles an *ADF*-instance into one program to solve the corresponding semantics.

Finally we want to introduce the DIAMOND-family<sup>21</sup> (**D**ialectical **M**odels **E**ncoding) of *ADF* solvers. It is the first *ADF*-solver done for the revisited version of *ADFs*, is still being developed further, and was prominent during the last years in the scientific community. DIAMOND is a set of static encodings to compute the various semantics of *ADFs*. One feature of them is that they implement the *AFT* directly and that each semantics encoding is a collection of different modular encoding-files. That allows one to easily extend the featureset of the tool like adding additional semantics, or using another operator for other theories. In fact DIAMOND has a distinct operator for Dung Argumentation Frameworks, *ADFs*, *BADFs*, and theorybases. For *BADFs* it also provides an option to compute the link-types beforehand. Due to the big number of small modular encodings which need to be used together to get the static encoding for one semantic, the system is also providing some sort of wrapping system.

---

<sup>18</sup><http://www.dbai.tuwien.ac.at/proj/adf/qadf>

<sup>19</sup><http://www.dbai.tuwien.ac.at/proj/adf/grappavis>

<sup>20</sup><http://www.dbai.tuwien.ac.at/proj/adf/yadf>

<sup>21</sup><http://diamond-adf.sourceforge.net>

The first versions of DIAMOND [Ellmauthaler and Strass, 2013, Ellmauthaler and Strass, 2014] used a PYTHON script to provide the needed usability. It has a streamlined set of command-line options to compute semantics. Later versions also allowed to answer the reasoning problems listed in the complexity subsection of Section 4.3.2. In addition some semantics (e.g. preferred) are realised by using two solver calls after another. This is for example done for the preferred semantics, where the first call computes all complete semantics and the second does the maximisation with respect to subsets. Instead the system also offers a disjunctive encoding to solve that in one call. DIAMOND 3.0 [Ellmauthaler and Strass, 2016], also called cDIAMOND has then been implemented in C++. The decision for that language was also influenced by the fact, that GRINGO and CLASP are realised in C++ too and that they provide a native library to incorporate the solver engine directly into ones application. Recently goDIAMOND [Strass and Ellmauthaler, 2017] has also been presented, which is another fork in the family, which uses the script language GO instead of PYTHON.

goDIAMOND recently participated in a competition about evaluating Dung Argumentation Frameworks. To compete with other solvers an approximated operator for Dung Frameworks has been used. Overall it seems that DIAMOND performed pretty well. So far two tracks are evaluated and in both goDIAMOND could claim the 4<sup>th</sup> place<sup>22</sup>.

---

<sup>22</sup>see <http://www.dbai.tuwien.ac.at/iccma17/results.html> for further information and the overall ranking



---

# Reactive Multi-Context Systems

---

In this chapter we will present the *reactive Multi-Context Systems*. They are a generalisation of Multi-Context Systems which have been presented in Section 3.4. Over the last years they got investigated and developed further. A first and rough concept was presented by me during a Workshop [Ellmauthaler, 2013]. There the reactivity which should offer integration and dynamics in a heterogeneous environment was called *iterative Multi-Context Systems*. Later that year the approach was also considered in an invited talk by Gerhard Brewka at the International Conference on Logic Programming and Nonmonotonic Reasoning [Brewka, 2013]. These first steps have been further developed and the first version of *reactive Multi-Context Systems* got introduced [Brewka et al., 2014a, Brewka et al., 2014b]. At the same time, a similar approach has been presented by the Lisbon University [Gonçalves et al., 2014], so called *evolving Multi-Context Systems*. The biggest difference between both systems has been that evolving Multi-Context Systems have a dedicated operator to change the Multi-Context System over time (i.e. the next-operator) and that the reactive Multi-Context Systems use a sequence called a run. The authors of both systems came to the conclusion that both approaches have their appeal and that it is preferable to combine both independent adaptations with each other. So the *reactive Multi-Context Systems* [Brewka et al., 2018] got revamped in a collaborative effort. We will first present the first and original version of *reactive Multi-Context Systems* and then we will introduce the revamped version to show how the formalism has developed during the last years<sup>23</sup>. Note that we do not unify the notation in between both adaptations, as the redefinition of many aspects has been one of the efforts and contributions regarding the evolution of reactive Multi-Context Systems.

---

<sup>23</sup>The original version is presented as a historical predecessor of the newer version, therefore the original version introduction will be fashioned in an overview-like style, while the new version will be investigated in a detailed manner

## 5.1 Original Reactive Multi-Context Systems

First, as for a managed Multi-Context System, we will need to define a context. For a simpler representation we have chosen to omit the *logic suite* (see Definition 3.4.1). The basic idea behind these logic suites is to allow different semantics to be used for the evaluation of a given knowledge base of a context. The decision for the option to have different kinds of semantics and limiting a context to only one has been done to make the already very technical notations easier to write. In addition it is generally possible to model different semantics via different representations in the knowledge base. It is possible to simulate that behaviour by using a more general semantics, which can be adjusted by additional beliefs which represent the utilised semantics too. In other words, it is straightforward to shift that semantics feature into the contexts semantics.

**Definition 5.1.1.** *A context is of the form  $C = \langle L, ops, mng \rangle$  where*

- $L = \langle KB_L, BS_L, \mathbf{acc}_L \rangle$  is a logic,
- $OP$  is a set of operations,
- $mng : 2^{OP} \times KB_L \rightarrow KB_L$  is a management function.

Basically one context consists of a logic, a set of operators to modify the knowledge base of the context, and a management function which implements the intended changes of the operators. The second adaption in contrast to the managed Multi-Context Systems is to assume that the management function is deterministic. Because we have restricted our contexts to only operate on one semantics the management function no longer needs to prepare different knowledge bases. That could have been needed, based on the different computations regarding semantics<sup>24</sup>. This case is no longer a priority and there are still sources of non-determinism, such that the flavour of the semantics and the Equilibrium will not be altered at all. Due to the changes one aspect of the function shifts: The management function has to handle how overlapping effects of two simultaneously applied operators will behave (e.g. addition and deletion of the same knowledge, or revision paired with any other operator). Beforehand it was not needed to warrant a definition of the order of application for the different operators, as every permutation could be deduced by the non-deterministic nature of the function. Still, this is only a minor issue, as it is still possible to emulate the non-deterministic behaviour, which will be shown later in Example 5.1.10.

To facilitate an information flow from the environment, we will need to specify a source of data which contains the continuous stream of readings. We assume that a sensor  $\Pi$  is a device which is able to provide new information in

---

<sup>24</sup>As an example a negation operator might need to differentiate between classical and non-classical negation based on the used semantics

a given language  $L_\Pi$  specific to the sensor. From an abstract point of view, we can identify a sensor with its observation language and a current sensor reading, that is  $\Pi = \langle L_\Pi, \pi \rangle$  where  $\pi \subseteq L_\Pi$ . Given a tuple of sensors  $\Pi = \langle \Pi_1, \dots, \Pi_k \rangle$ , an observation  $Obs$  for  $\Pi$  ( $\Pi$ -observation for short) consists of a sensor reading for each sensor, that is  $Obs = \langle \pi_1, \dots, \pi_k \rangle$  where for  $1 \leq i \leq k$ ,  $\pi_i \subseteq L_{\Pi_i}$ . Note that such a reading represents only a slice of the data at a given point in time and does not facilitate any continuous properties of changing information over time.

We can now define a reactive Multi-Context System over sensors, where we represent the connectivity between the contexts and the sensor input in a static and noncontinuous way.

**Definition 5.1.2.** *A reactive Multi-Context System over a tuple of sensors  $\Pi = \langle \Pi_1, \dots, \Pi_k \rangle$  is a triple  $M = \langle C, BR, KB \rangle$  consisting of:*

- a tuple of Contexts  $C = \langle C_1, \dots, C_n \rangle$ ,
- a tuple  $BR = \langle BR_1, \dots, BR_n \rangle$ , where each  $BR_i$  is a set of bridge rules for  $C_i$  over  $C$ . A bridge rule for  $C_i$  over  $C$  is of the form

$$op \leftarrow a_1, \dots, a_j, \text{not } a_{j+1}, \dots, \text{not } a_m,$$

such that  $op \in OP_i$  and every  $a_l (1 \leq l \leq m)$  is either an atom of form  $c : b$ , where  $c \in \{1, \dots, n\}$ , and  $b$  is a belief for  $C_c$ , i.e.  $b \in B$  for some  $B \in BS_c$  or a sensor atom in the form  $o@s$ , where  $s$  is an index determining a sensor ( $1 \leq s \leq k$ ) and  $o \in L_{\Pi_s}$  is some sensordata, and

- a tuple of Knowledge bases  $KB = \langle kb_1, \dots, kb_n \rangle$ , such that  $kb_i \in KB_{L_i}$ .

Intuitively, we extend the scope of a managed Multi-Context System, such that each bridge rule may either refer to the beliefs of a given context, or the information provided by a sensor. We have chosen to use  $c : b$  to describe that  $b$  is believed in  $C_c$ . To stay intuitive to the meaning of the @-Symbol, we use  $a@s$  to represent that  $a$  occurs in sensor  $s$ , such that it reads as “a at s”. In addition the reactive Multi-Context System defines its initial set of knowledge bases for each context. Again, in contrast to the managed Multi-Context System, here the interconnection of the different contexts is managed by the multi-context system, while in the managed version each “managed context” encapsulates this information. This change is to reduce the number of indices for convenience, as the bridge rules and later the equilibrium notion will need access to the initial knowledge bases.

The combined view on the belief-states of all contexts in a reactive Multi-Context System is called *belief state*.

**Definition 5.1.3.** *Let  $M = \langle C, BR, KB \rangle$  be a reactive Multi-Context System. We call  $B = \langle b_1, \dots, b_n \rangle$  a belief state of  $M$  if for each  $1 \leq i \leq n$ ,*

$$B_i \in BS_{L_i}, \text{ and } BS_{L_i} \in L_i \in C_i$$

The intended semantics for a bridge rule is to be *applicable* with respect to a belief state and sensors, if every proposition about the sensor data is consistent with the sensors. In addition the bridge rule still needs to satisfy the beliefs from the belief state, as it has been necessary for the managed Multi-Context System case before.

**Definition 5.1.4.** *Let  $\Pi$  be a tuple of sensors and  $Obs = \langle \pi_1, \dots, \pi_k \rangle$  a  $\Pi$ -observation. A sensor atom  $o@s$  is satisfied by  $Obs$  if  $o \in \pi_s$ ; a negated literal not  $o@s$  is satisfied by  $Obs$  if  $o \notin \pi_s$ .*

*Let  $M = \langle C, BR, KB \rangle$  be an rMCS with sensors  $\Pi$  and  $B$  a belief state for  $M$ . A bridge rule  $r$  in  $BR$  is applicable wrt.  $B$  and  $Obs$ , symbolically  $B \models_{Obs} body(r)$ , if every context literal in  $body(r)$  is satisfied by  $B$  and every sensor literal in  $body(r)$  is satisfied by  $Obs$ .*

For easier representation we use  $app_i(B, Obs) = \{head(r) \mid r \in br_i \wedge B \models_{Obs} body(r)\}$  to have a set of all operators which have been invoked by the bridge rules of a given context with respect to a belief-state and the observations of the sensors. It is now straightforward to define an equilibrium of an rMCS in a similar way as for an mMCS:

**Definition 5.1.5.** *Let  $M = \langle C, BR, KB \rangle$  be an rMCS with sensors  $\Pi$  and  $Obs$  a  $\Pi$ -observation. A belief state  $B = \langle b_1, \dots, b_n \rangle$  for  $M$  is an equilibrium of  $M$  under  $Obs$  if, for  $1 \leq i \leq n$ ,*

$$b_i \in ACC_i(mng_i(app_i(B, Obs), kb_i)).$$

In other words, some belief state is an equilibrium if the beliefs and the sensor data invoke such changes to the current knowledge base, such that the elaborated changes will lead to a state where the semantics of the contexts come to the same beliefs which have been assumed beforehand. It should be easy to see, that observations can be seen as some kind of static, unchangeable facts, while the belief state has a non-deterministic flavour, where the beliefs to reach need to be supported by the applicable bridge rules and some sort of agreement of the context semantics.

Till now we have extended the managed Multi-Context Systems to a point where they are aware of sensor input, which needs to be seen as some sort of real-world facts. To get towards reactivity to continuous computation, reasoning and awareness on sensor data, we need to keep track of the changing knowledge bases. For that goal we pair an equilibrium together with the updated knowledge base which got induced by the management function and the applicable bridge rules.

**Definition 5.1.6.** *Let  $M = \langle C, BR, KB \rangle$  be an rMCS with sensors  $\Pi$ ,  $Obs$  a  $\Pi$ -observation, and  $B = \langle b_1, \dots, b_n \rangle$  an equilibrium of  $M$  under  $Obs$ .  $KB^B = \langle mng_1(app_1(B, Obs), kb_1), \dots, mng_n(app_n(B, Obs), kb_n) \rangle$  is the tuple of KBs generated by  $B$  and  $Obs$ . The pair  $\langle B, KB^B \rangle$  is called full equilibrium of  $M$  under  $Obs$ .*



The relation between the equilibrium and the newly constructed knowledge bases now allows us to define a sequence of full equilibria. We now introduce the notion of a run of an rMCS induced by a sequence of observations. This sequence represents the flow of incoming information from sensor input over time.

**Definition 5.1.7.** *Let  $M = \langle C, BR, KB \rangle$  be an rMCS with sensors  $\Pi$  and  $O = (Obs^0, Obs^1, \dots)$  a sequence of  $\Pi$ -observations. A run of  $M$  induced by  $O$  is a sequence of pairs  $R = (\langle B^0, KB^0 \rangle, \langle B^1, KB^1 \rangle, \dots)$  such that*

- $\langle B^0, KB^0 \rangle$  is a full equilibrium of  $M$  under  $Obs^0$ ,
- for  $\langle B^i, KB^i \rangle$  with  $i > 0$ ,  $\langle B^i, KB^i \rangle$  is a full equilibrium of  $\langle C, BRs, KB^{i-1} \rangle$  under  $Obs^i$ .

### Modelling and Discussion

Following the definitions of the original reactive Multi-Context System we will now illustrate how different basic modelling features can be implemented. In addition we are also interested in discussing different difficulties or needed workarounds to make some things working. We are interested in showing how the mechanics of the interconnection of belief states, observations, and the run are working together. Therefore very minimal examples will be shown. For easy to understand toy environments we will stick to answer set programming contexts, as they are under a non-monotonic semantics with good expressiveness. Additionally it is straightforward how to manipulate a given logic program with respect to basic operators like addition or deletion.

**Example 5.1.8.** *An illustration on how an answer set program can be represented in the notion of a logic, as it is used for contexts is as follows. To capture the logic  $L_{asp} = \langle KB_{asp}, BS_{asp}, \mathbf{acc}_{asp} \rangle$  for answer set programs we need the set of all ground (i.e. variable free) atoms  $A$  which might appear in an answer set program.  $KB_{asp}$  is the set of all answer set programs over  $A$ , the possible belief states  $BS_{asp} = 2^A$  is given by the set of all possible answer sets, and  $\mathbf{acc}_{asp}$  maps each answer set program to the set of its answer sets.*

One field to explore in relation with multi-context systems is to understand how the non-determinism of bridge rules is used and applied. In contrast to the similar approach of answer set programming, the negation of literals is not the source to consider the concept of non-deterministic choices.

**Example 5.1.9.** *Assume a reactive Multi-Context System with two contexts  $C_1$  and  $C_2$ . Both are answer set programming contexts and the initial knowledge bases are as follows:  $KB = \langle \{a \leftarrow\}, \{b \leftarrow\} \rangle$ . Both contexts get the operations  $add(X)$ , and the semantics are defined by the management function such that for every  $OP'_c \subseteq OP_c$  and every  $x \in A$  (as defined as in Example 5.1.8) it holds*

that  $mng(OP', kb) = kb \cup \{x \leftarrow \mid add(x) \in OP'\}$ . Now consider the following bridge rule for  $BR_1 = \{add(b) \leftarrow 2 : b\}$ . In this case the only equilibrium would be  $B_1 = \langle \{a, b\}, \{b\} \rangle$ .

In addition we will now consider another bridge rule for the second context:  $BR_2 = \{add(a) \leftarrow 2 : a\}$ . If we check the previous equilibrium, we will see that  $B_1$  is an equilibrium for the reactive Multi-Context System too. This time we get a second equilibrium  $B_2 = \langle \{a, b\}, \{a, b\} \rangle$ .

By adding the rule  $\{del(a) \leftarrow not\ 1 : a\}$  to the rules of  $BR_1$ , and by enhancing the management function to work with a delete operator too, such that  $mng(OP', kb) = kb \cup \{x \leftarrow \mid add(x) \in OP'\} \setminus \{x \leftarrow \mid del(x) \in OP'\}$ , we get a similar behaviour: The belief sets are as follows:

$$BS = \{ \langle \{b\}, \{b\} \rangle, \langle \{a, b\}, \{b\} \rangle, \langle \{b\}, \{a, b\} \rangle, \langle \{a, b\}, \{a, b\} \rangle \}$$

The above example illustrates that notions of cyclic *self-support* and *self-negation* both lead to non-deterministic behaviour and have a nonmonotone flavour. Note that these cyclic constructs may be a reason for inconsistent rules, if they are combined such that existence will imply the removal of the information from the belief set or vice versa. Applications of rules, which use only sensor data, or where the operation is not in direct relation to the used beliefs in the body, are not considered to have non-deterministic features.

With these different ways to utilise non-deterministic rules, we want to show how to model the non-deterministic behaviour of the managed Multi-Context System management functions.

**Example 5.1.10.** Consider a reactive Multi-Context System with three answer set programming contexts  $C_1, C_2$ , and  $C_3$  with the initial knowledge bases  $KB = \langle \{a \leftarrow\}, \{b \leftarrow\}, \{b \leftarrow\} \rangle$ . The bridge rules for  $C_1$  are  $BR_1 = \{add(b) \leftarrow 2 : b, del(b) \leftarrow 3 : b\}$ .

In words, we have the rule that if  $b$  is believed in context  $C_2$ , then  $C_1$  should add  $b$  to its knowledge base. The believe of  $b$  in context  $C_3$  on the other hand will lead to the deletion of  $b$  in  $C_1$ . The example leads us to the point where it is not clear whether addition or deletion should be done first. How this is handled is defined in the management function. Two different possibilities for such a management function will be:

$$\begin{aligned} mng_1(OP', kb) &= (kb \cup \{x \leftarrow \mid add(x) \in OP'\}) \setminus \{x \leftarrow \mid del(x) \in OP'\} \\ mng'_1(OP', kb) &= (kb \setminus \{x \leftarrow \mid del(x) \in OP'\}) \cup \{x \leftarrow \mid add(x) \in OP'\} \end{aligned}$$

Depending on the used management function the behaviour might be different if both operators are deducted by the bridge rules. In the non-deterministic version both options are possible. To simulate this we can introduce two new bridge rules for  $C_1$  :

$$\begin{aligned} add(deloveradd) &\leftarrow 1 : deloveradd, \text{ and} \\ pref(del) &\leftarrow 1 : deloveradd \end{aligned}$$

Note that this atom *deloveradd* needs to be a new atom, which does not interfere with the reasoning of  $C_1$  and is not occurring in another bridge rule. Then the management function needs to be adapted too:

$$\begin{aligned} \text{mng}_1(OP', kb) &= \\ &= \begin{cases} (kb \cup \{x \leftarrow \mid \text{add}(x) \in OP'\}) \setminus \{x \leftarrow \mid \text{del}(x) \in OP'\} & \text{if } \text{pref}(\text{del}) \in OP' \\ (kb \setminus \{x \leftarrow \mid \text{del}(x) \in OP'\}) \cup \{x \leftarrow \mid \text{add}(x) \in OP'\} & \text{otherwise} \end{cases} \end{aligned}$$

Through the two different belief sets which might be considered to be an equilibrium - the one where *deloveradd* is considered to be true and the one where it is not - we can simulate that non-deterministic flavour of applying both combinations of the operators.

To give the decision of possible preferences for the order of operators and the specific implementation of the operators in the management function to the modelling layer has been one additional motivation for using the non-deterministic management function. Note that in many cases it might not be desirable to allow every combination of operators. Sometimes it might even be counter intuitive to allow some combinations (e.g. doing repairs to ensure consistency of a given context should be done after all other operations on the context, not beforehand). In addition there might not be a need for a specific order of events. Example 5.1.10 had a setup where two contradicting operators are used on the same knowledge. If different operators are not contradicting each other, their order might not be an issue at all.

When reasoning is done over time, one might be interested in taking time into account. This can be done in an easy way, by having a sensor, which just provides the current time. Another option, as we have a discrete sequence of sensor data which induces the discrete run, would be to use the same numbering as the indexes for the sensor data observations. We will refer to that specific point in time as a timestamp. To utilise such a numbering we have different options to consider:

- Use a sensor, which observes the current timestamp,
- use a built-in command, which is used as a constant term, which is just substituted with the current number, or
- utilise a context to model that behaviour.

While all three options are viable, we would like to have a solution where we don't need to add additional mechanics to the reactive Multi-Context System. Therefore we are not interested in exploring the second option further. Utilising a sensor observation is intuitive and trivial to use. So we will now investigate how to model the behaviour with a context.

**Example 5.1.11.** *The context  $C_1$  is a timing context, with exactly one fact, such that  $\text{KB} = \{0 \leftarrow\}$ . To implement such a step-timer we don't need to use*

*any additional bridge rules, but can use the management function as follows:*

$$mng_1(OP', kb) = \{x + 1 \leftarrow \mid x \leftarrow \in kb\}$$

Note that it is not needed to use bridge rules at all, as this operation should be done during every step in a run. In some sense it can be seen as some operation which is applied every time, independent of beliefs or knowledge of other contexts or the context itself. This shows the versatility of this framework, as it allows to reason over beliefs and sensor information in a global way, while the management function allows to change the knowledge bases based on the operators, as well as on basis of its own knowledge base, and in a static way, without any dependencies at all.

In general we have discussed how to model some simple concepts with reactive Multi-Context Systems. Modelling of things like forgetting (i.e. removing information based on some knowledge or beliefs) or flipping values seems easy and straightforward. Alas it gets more complicated if we want that the forgettable or switched knowledge is part of the reasoning for doing that. An easy example for such a scenario might be that some switches might be turned on by some external sensor data. As a security protocol it should be enforced that if all switches are turned on, one should be flipped off. To model such an occurrence we will need to introduce an intermediate step, like adding some knowledge such that the alert state has been triggered. In the next step then the switch can be flipped backward because of the alert state, but not because the switch itself is turned on. This kind of intermediate steps and a high amount of cleanup actions to get rid of these alert states, makes this original version of a reactive Multi-Context System pretty complicated.

## 5.2 Reactive Multi-Context Systems

Due to the previously presented shortcomings of the original reactive Multi-Context Systems, which has been mainly the inability to let bridge rules change values of the knowledge base justified by the values itself, and the quite complicated formalism, a complete rework of reactive Multi-Context Systems has been done. This rework has been aiming towards an easier and straightforward definition of concepts and the integration of the next-operator, introduced by the research group in Lisbon. In addition it was our goal to unify the two similar approaches from Lisbon [Gonçalves et al., 2014] and Leipzig [Brewka et al., 2014b] in a cooperative effort.

First we simplified our formalism by creating a uniform convention when writing symbols in formal sections. That increases readability and makes it easier to follow the concepts on a natural understanding of the different components. We will use lower-case words to refer to single entities. Sets of entities and structures with different components are in upper-case, while sequences are written in **sans serif**. Temporal dimensions are upper-case only and use calligraphic letters, like  $\mathcal{S}$  or  $\mathcal{I}$ . Additionally, we will use **bold** letters to denote functions and operators. Finally, to denote specific elements in examples, we will use **typewriter** fonts.

We are still using the management function and bridge rules, to specify the flow of information between contexts, which is additionally the way to allow conflict resolution in case of contradictory context knowledge. The presentation of this topic will be with original reactive Multi-Context Systems in mind, but we will not discuss every small difference in a comparative manner.

In the next section we will discuss the different components of reactive Multi-Context Systems. That means we will stick to the syntax and present the intuitive ideas behind the different concepts. Afterwards, in Section 5.2.2, we will present the precise and formal semantics of reactive Multi-Context Systems. Then we will discuss how to model the assisted living example scenario from the Motivation (cf. Chapter 2) in Section 5.2.3. In addition we will have some considerations about variables in bridge rules too. Based on all these considerations a discussion about different modelling concepts will be presented. Section 5.2.4 is dedicated to the discussion of inconsistencies, how to avoid them, and how to manage situations where they cannot be avoided. Then Section 5.2.5 will discuss how to react if the inverse case happens, namely that too many possible belief sets are valid equilibria. In the following section we will show the expressiveness of reactive Multi-Context System by presenting a way to simulate a turing machine by only using non-expressive context logic. Finally, in Section 5.2.7, we will discuss the computational complexity of the presented theory.

### 5.2.1 Components of a Reactive Multi-Context System

Like before, a context will be built upon the abstract notion of a logic. To build a heterogeneous scenery of contexts, we will not only stick to answer set programming settings, as we have done it in Section 5.1.

**Example 5.2.1.** *We illustrate how different formalisms are used and how they can be represented by the notion of a logic.*

To capture the description logic  $\mathcal{AL}$  as  $L_d = \langle KB_d, BS_d, \mathbf{acc}_d \rangle$ , we consider  $KB_d$  as the set of all well-formed description logic knowledge bases over  $\mathcal{AL}$ , which are all ontologies.  $BS_d$  is the set of deductively closed subsets in  $\mathcal{AL}$  and  $\mathbf{acc}_d$  is a mapping which maps each  $kb \in KB_d$  to  $\{E\}$ , where  $E$  is the set of formulas in  $\mathcal{AL}$ , such that  $kb \models E$ .

Given a set of  $E$  of entries, we can construct a simple storage logic to store elements from  $E$  by the logic  $L_s = \langle KB_s, BS_s, \mathbf{acc}_s \rangle$ , such that  $KB_s = BS_s = 2^E$  and  $\mathbf{acc}_s$  maps every set  $E' \subseteq E$  to  $\{E'\}$ . This is an easy way to use logic, to represent a simple version of a database logic. Further on we will call this logic a storage logic.

In Example 5.1.8, we have already presented how to realise a logic programming logic by instantiating the answer set programming paradigm in terms of the concept of an abstract logic.

**Definition 5.2.2** (Context). *A context is a triple  $C = \langle L, OP, \mathbf{mng} \rangle$  where*

- $L = \langle KB, BS, \mathbf{acc} \rangle$  is a logic,
- $OP$  is a set of operations,
- $\mathbf{mng} : 2^{OP} \times KB \rightarrow KB$  is a management function.

As for the original version we use the context in the same way. For an indexed context  $C_i$  we will write  $L_i = \langle KB_i, BS_i, \mathbf{acc}_i \rangle$ ,  $OP_i$ , and  $\mathbf{mng}_i$  to denote its contexts. Operations are as before just labels for intended semantics, which will be carried out and computed by the management function.

**Example 5.2.3.** *We will now consider the assisted living example scenario from the Motivation chapter (cf. Chapter 2). In that case we want to detect and recognise different potential threats to the inhabitant which may be caused by items in the flat. One such source of a threat can be the stove (e.g. overheating, inflaming leftover stuff if forgotten to turn off,...).*

First we will use a stove monitor context  $C_{st}$ . For that easy monitor we will utilise our simple storage logic from Example 5.2.1.  $C_{st}$ 's logic  $L_{st}$  is a storage logic, using  $E = \{pw, tm(cold), tm(hot)\}$  as the set of possible entries.  $pw$  will denote whether the power is on or off. If the entry is existing, the power is on, and off otherwise. The temperature  $tm$  on the other hand has a qualitative value, to reflect that the stove is either **cold** or **hot**.

The context keeps track of the current state of the stove by having the corresponding entries in its knowledge base. To manipulate the value, the following operations are used:

$$OP_{st} = \{setPower(off), setPower(on), setTemp(cold), setTemp(hot)\}$$

The semantics for a set of operators  $OP' \subseteq OP_{st}$  is given by

$$\begin{aligned} \mathbf{mng}_{st}(OP', kb) = & \{pw \mid setPower(on) \in OP' \vee \\ & (pw \in kb \wedge setPower(off) \notin OP')\} \cup \\ & \{tm(t) \mid setTemp(t) \in OP'\}. \end{aligned}$$

With these semantic definitions we assume that there is a temperature sensor, which will provide the context constantly with the given temperature of the stove. It will trigger all the time the current qualitative value for the `setTemp` operation. The operator is implemented in such a way that the given value will just be inserted with every management function call and we do not need to care about the possibility of conflicting information or non-persistent temperature values. For the power switch we are thinking about a switch, which will toggle the stove to be on or off. We consider the stove to be on if there is no information about it being off. In addition the fluent `pw` will stay on if it is on and not turned off. `pw` can only be deduced if it is already on and not turned off by the corresponding operator, or if it is turned on by the matching operator. Note that due to the formalisation of the management function, that in case of conflicting information about the switch (i.e. `setPower(on)` and `setPower(off)`) are both in  $OP'$ , then the stove is considered to be on. That is a distinct choice, because it might be much more harmful if the stove stays on unattended than sounding an alarm or turning off the electricity for the stove unnecessarily.

To illustrate how the management function will work, we assume the knowledge base  $kb = \{tm(cold)\}$ , together with the assumed set of operators  $OP = \{setPower(on), setTemp(hot)\}$ . The update on the knowledge base with respect to the given operations is  $\mathbf{mng}_{st}(OP, kb) = \{pw, tm(hot)\}$ .

Next we will need to introduce the bridge rules. They allow communication between different contexts and as an important core mechanic of Multi-Context Systems, we will define them in a stand-alone way which is not directly embedded to the Multi-Context System. As bridge rules are an abstract approach to model the exchange of beliefs between different contexts and integrate information from the outside world, we will need to have a distinct understanding of this outside world. To keep it simple, but still expressive, we will require inputs from the outside world to be elements of some formal language  $IL$ . In addition we want to differentiate between different sources of input (i.e. different sensors if spoken from the point of view of original reactive Multi-Context

Systems). That will be done by speaking of a tuple of different input languages  $\mathbb{IL} = \langle IL_1, \dots, IL_k \rangle$  which are considered to be associated with the different outside world sources.

**Definition 5.2.4** (Bridge Rule). *Let  $\mathbb{C} = \langle C_1, \dots, C_n \rangle$  be a tuple of contexts and  $\mathbb{IL} = \langle IL_1, \dots, IL_k \rangle$  a tuple of input languages. A bridge rule for  $C_i$  over  $\mathbb{C}$  and  $\mathbb{IL}$ ,  $i \in \{1, \dots, n\}$ , is of the form*

$$\mathbf{op} \leftarrow a_1, \dots, a_j, \mathbf{not} a_{j+1}, \dots, \mathbf{not} a_m \quad (5.1)$$

such that  $\mathbf{op} = op$  or  $\mathbf{op} = \mathbf{next}(op)$  for  $op \in OP_i$ ,  $j \in \{0, \dots, m\}$ , and every atom  $a_\ell$ ,  $\ell \in \{1, \dots, m\}$ , is one of the following:

- a context atom  $c:b$  with  $c \in \{1, \dots, n\}$  and  $b \in B$  for some  $B \in BS_c$ , or
- an input atom  $s::b$  with  $s \in \{1, \dots, k\}$  and  $b \in IL_s$ .

For a bridge rule  $r$  of the form (5.1)  $\text{head}(r)$  denotes  $\mathbf{op}$ , the head of  $r$ , while  $\text{body}(r) = \{a_1, \dots, a_j, \mathbf{not} a_{j+1}, \dots, \mathbf{not} a_m\}$  is the body of  $r$ . A literal is either an atom or an atom preceded by  $\mathbf{not}$ , and we differentiate between context literals and input literals.

Intuitively, these bridge rules will still consider some abstract operation to be executed on the different knowledge bases of contexts based on some beliefs. In addition to the already known concept, we are now defining them only for different contexts and input languages, totally unrelated to a reactive Multi-Context System. We have chosen to represent input atoms with the same intuition as context atoms: With a different symbol but the same intended semantics. Note that this way we avoid using unintended meaning of some letters (i.e. the "@"-Symbol). Here we introduce the intuition of the next operator too. In simple words we are going to differentiate between temporary operators used for the computation of the equilibrium only (i.e. those which are without next) and those which will be used to permanently change the knowledge bases (i.e. the next-operations). This is just a intuitive primer and the precise semantics will be presented later in Section 5.2.2.

**Definition 5.2.5** (Reactive Multi-Context System). *A reactive Multi-Context System is a tuple  $M = \langle \mathbb{C}, \mathbb{IL}, \mathbf{BR} \rangle$ , where*

- $\mathbb{C} = \langle C_1, \dots, C_n \rangle$  is a tuple of contexts;
- $\mathbb{IL} = \langle IL_1, \dots, IL_k \rangle$  is a tuple of input languages;
- $\mathbf{BR} = \langle BR_1, \dots, BR_n \rangle$  is a tuple such that each  $BR_i$ ,  $i \in \{1, \dots, n\}$ , is a set of bridge rules for  $C_i$  over  $\mathbb{C}$  and  $\mathbb{IL}$ .

The final syntactic definition of a reactive Multi-Context System is now pretty easy to read and understand. It is just a set of contexts and input languages, connected with the newly introduced bridge rules.



**Example 5.2.6.** We are using the reactive Multi-Context System  $M_{exa} = \langle \langle C_{st} \rangle, \langle IL_{exa} \rangle, \langle BR_{exa} \rangle \rangle$  which is a Multi-Context System utilising the context  $C_{st}$  from Example 5.2.3. Note that we will use the names of contexts (e.g.  $C_{st}$ ) instead of its numerical indices for better readability. These are just labels and do not extend the concept of the presented indices. We will now show how to define bridge rules with the **next** operator to implement the switch behaviour discussed before for the stove:  $IL_{exa} = \{switch\}$  is used to report the outside world event of whether the switch of the stove has been pushed or not. The corresponding bridge rules in  $BR_{exa}$  are

$$\begin{aligned} \mathbf{next}(setPower(on)) &\leftarrow exa::switch, \mathbf{not} \ st:pw \text{ and} \\ \mathbf{next}(setPower(off)) &\leftarrow exa::switch, st:pw. \end{aligned}$$

It is easy to understand what is meant by these two bridge rules. Both can only be triggered if there is some input from the outside, which states that the switch has been pushed. Then, depending on the current belief of the stove context, the knowledge of the context about the power state of the stove is set permanently.<sup>25</sup> Note that this is an application of the **next**-operator, where we really need it. As  $setPower$  will change the knowledge on  $st:pw$ , the cyclic dependency from the discussion on the original reactive Multi-Context Systems is in place.

## 5.2.2 Semantics of Reactive Multi-Context System

The definition of the semantics of reactive Multi-Context Systems will be done step by step. Which means we will first define the different pieces and then puzzle them together. First, only the static case with the computation of one single time instant will be shown. There we will discuss and show how the bridge rules are evaluated and how the equilibria are computed. After the interplay between **next** operators and non-**next** operators is pictured, we will advance to the subsequent introduction of the dynamic notions, where the system may react to changes over time.

To evaluate a bridge rule, we need to know how the current state of beliefs for each context is. This is pooled together as a *belief state*.

**Definition 5.2.7** (Belief State). *Let  $M = \langle \langle C_1, \dots, C_n \rangle, IL, BR \rangle$  be a reactive Multi-Context System. Then, a belief state for  $M$  is a tuple  $B = \langle B_1, \dots, B_n \rangle$  such that  $B_i \in BS_i$ , for each  $i \in \{1, \dots, n\}$ . We use  $\mathbf{Bel}_M$  to denote the set of all belief states for  $M$ .*

To capture the current external information too, we introduce the notion of an *input*. Note that input and belief sets follow the same basic concept, once for the contexts and once for the external information sources.

<sup>25</sup>We use permanently in a deliberate manner to underline the difference between **next** and “normal” operators. It should be intuitively clear that the knowledge base may change again at the next computation step

**Definition 5.2.8** (Input). *Let  $M = \langle C, \langle IL_1, \dots, IL_k \rangle, BR \rangle$  be a reactive Multi-Context System. Then an input for  $M$  is a tuple  $I = \langle I_1, \dots, I_k \rangle$  such that  $I_i \subseteq IL_i$ ,  $i \in \{1, \dots, k\}$ . The set of all inputs for  $M$  is denoted by  $\text{Inp}_M$ .*

With a distinct overview of the current state of beliefs for the contexts and the information provided by the external sources, we can investigate the semantics of bridge rules. Intuitively we want a bridge rule to be *applicable* if the belief sets and external information are satisfying every literal in the body of the bridge rule.

**Definition 5.2.9** (Satisfaction of Literals). *Let  $M = \langle C, IL, BR \rangle$  be a reactive Multi-Context System, such that  $C = \langle C_1, \dots, C_n \rangle$  and  $IL = \langle IL_1, \dots, IL_k \rangle$ . Given an input  $I = \langle I_1, \dots, I_k \rangle$  for  $M$  and a belief state  $B = \langle B_1, \dots, B_n \rangle$  for  $M$ , we define the satisfaction of literals as:*

- $\langle I, B \rangle \models a_\ell$  if  $a_\ell$  is of the form  $c:b$  and  $b \in B_c$ ;
- $\langle I, B \rangle \models a_\ell$  if  $a_\ell$  is of the form  $s:b$  and  $b \in I_s$ ;
- $\langle I, B \rangle \models \mathbf{not} a_\ell$  if  $\langle I, B \rangle \not\models a_\ell$ .

Let  $r$  be a bridge rule for  $C_i$  over  $C$  and  $IL$ . Then

- $\langle I, B \rangle \models \text{body}(r)$  if  $\langle I, B \rangle \models l$  for every  $l \in \text{body}(r)$ .

We will say that a bridge rule is applicable under  $\langle I, B \rangle$  if and only if  $\langle I, B \rangle \models \text{body}(r)$  holds. It is now straight forward that we want to know which operators are applicable, based on the bridge rules.

**Definition 5.2.10** (Applicable Operators). *Let  $M = \langle C, IL, BR \rangle$  be a reactive Multi-Context System, such that  $C = \langle C_1, \dots, C_n \rangle$  and  $BR = \langle BR_1, \dots, BR_n \rangle$ . Given an input  $I$  for  $M$  and a belief state  $B$  for  $M$ , we define, for each  $i \in \{1, \dots, n\}$ , the sets*

- $\mathbf{app}_i^{\text{now}}(I, B) = \{\text{head}(r) \mid r \in BR_i, \langle I, B \rangle \models \text{body}(r), \text{head}(r) \in OP_i\}$ ;
- $\mathbf{app}_i^{\text{next}}(I, B) = \{op \mid r \in BR_i, \langle I, B \rangle \models \text{body}(r), \text{head}(r) = \mathbf{next}(op)\}$ .

Here we can see that we strictly distinguish between the **next**-operator and the others. As already hinted before, we are going to have temporary operators, which will be used for volatile changes on the knowledge bases for the semantics during one time step. For the transition and permanent change of information between two time steps, we will use the next operators.

The thoughtful reader might have perceived that in the new definition of reactive Multi-Context Systems, we have not defined the current knowledge bases as part of the Multi-Context System. The continuous change of knowledge bases over time and the interplay between volatile and permanent changes on knowledge bases is the reason we consider the knowledge bases in another

way. To associate distinct knowledge bases to the contexts of a reactive Multi-Context System, we use the concept of a *configuration* structure to store them as additional information. In addition this approach is more modular and might allow future changes or tweaks for different components without the need to change the whole system.

**Definition 5.2.11** (Configuration of Knowledge Bases). *Let  $M = \langle C, \text{IL}, \text{BR} \rangle$  be a reactive Multi-Context System, such that  $C = \langle C_1, \dots, C_n \rangle$ . A configuration of knowledge bases for  $M$  is a tuple  $\text{KB} = \langle kb_1, \dots, kb_n \rangle$ , such that  $kb_i \in \text{KB}_i$ , for each  $i \in \{1, \dots, n\}$ . We use  $\text{Con}_M$  to denote the set of all configurations of knowledge bases for  $M$ .*

With all the different pieces from this section, we have all information we will need to compute a meaningful semantics, which is faithful to the basic concepts of equilibria from previous Multi-Context Systems. The semantics for a reactive Multi-Context System for a single time instant is given as an *equilibrium*.

**Definition 5.2.12** (Equilibrium). *Let  $M = \langle \langle C_1, \dots, C_n \rangle, \text{IL}, \text{BR} \rangle$  be a reactive Multi-Context System,  $\text{KB} = \langle kb_1, \dots, kb_n \rangle$  a configuration of knowledge bases for  $M$ , and  $l$  an input for  $M$ . Then, a belief state  $\text{B} = \langle B_1, \dots, B_n \rangle$  for  $M$  is an equilibrium of  $M$  given  $\text{KB}$  and  $l$  if, for each  $i \in \{1, \dots, n\}$ , we have that*

$$B_i \in \text{acc}_i(kb'), \text{ where } kb' = \text{mng}_i(\text{app}_i^{\text{now}}(l, \text{B}), kb_i).$$

Basically, the definition of an equilibrium does not change at all. We are still using the applicable operators from all the rules without the **next**-operator and apply the management function on the knowledge base. If the newly computed knowledge base has the belief set as an accepted result of its semantics, the belief set is considered to be an equilibrium.

**Example 5.2.13.** *We will use the reactive Multi-Context System  $M_{\text{exa}}$  from Example 5.2.6. Consider the configuration of knowledge bases for  $M_{\text{exa}}$  as  $\text{KB} = \langle kb_{\text{st}} \rangle = \langle \emptyset \rangle$ , an input  $l = \langle \{\text{switch}\} \rangle$ , and the belief state  $\text{B} = \langle \emptyset \rangle$ . This represents that the stove is currently off, since the current configuration has not **pw** in the knowledge base of the stove context. In addition the input represents that the switch has been pressed and the belief set is considering that nothing has changed so far. As the bridge rules  $\text{BR}_{\text{exa}}$  only use **next**-operators in the heads,  $\text{app}^{\text{now}}l, \text{B} = \emptyset$  are the applicable operators and following the definition of  $\text{mng}_{\text{st}}$  from Example 5.2.3, the knowledge base for the stove context does not changes (i.e.  $\text{mng}_{\text{st}}(\text{app}^{\text{now}}(l, \text{B}), kb_{\text{st}}) = kb_{\text{st}}$ ) Therefore  $\text{acc}_{\text{st}}(\text{mng}_{\text{st}}(\text{app}^{\text{now}}(l, \text{B}), kb_{\text{st}})) = \{\emptyset\}$ . Thus,  $\text{B}$  is an equilibrium of  $M_{\text{exa}}$  given  $\text{KB}$  and  $l$ .*

Now we are still missing how to make the permanent changes to the knowledge base for the transition to get a new configuration after an equilibrium

is computed. Basically, we are utilising an *update function* to produce a new configuration of knowledge bases to emphasise the updates which are concluded by a given belief state.

**Definition 5.2.14** (Update Function). *Let  $M = \langle C, \mathbb{L}, \mathbb{B}R \rangle$  be a reactive Multi-Context System such that  $C = \langle C_1, \dots, C_n \rangle$ ,  $\mathbb{K}B = \langle kb_1, \dots, kb_n \rangle$  a configuration of knowledge bases for  $M$ ,  $I$  an input for  $M$ , and  $B$  a belief state for  $M$ .*

*Then,  $\mathbf{upd}_M(\mathbb{K}B, I, B) = \langle kb'_1, \dots, kb'_n \rangle$  is the update function for  $M$ , such that for each  $i \in \{1 \dots, n\}$ ,  $kb'_i = \mathbf{mng}_i(\mathbf{app}_i^{next}(I, B), kb_i)$  holds.*

Note that we define the update function on the belief state and not an equilibrium. The connection between equilibrium and update function will be done in the following by the investigation of the role of reactive Multi-Context Systems in the dynamic setting over time. To refer to different time instances, we will utilise the idea of logical time instants. That means we label each time instance, where we will do reasoning, with a natural number. These logical instances are not representing physical time points and we do not require that pairs of consecutive natural numbers represent equidistant physical time spans. Still, we do require that every natural number is utilised as a cardinal number (i.e. a (infinite) sequence starting with 1 and a step size of 1).

**Definition 5.2.15** (Input Stream). *Let  $M = \langle C, \mathbb{L}, \mathbb{B}R \rangle$  be a reactive Multi-Context System such that  $\mathbb{L} = \langle IL_1, \dots, IL_k \rangle$ . An input stream for  $M$  (until  $\tau$ ) is a function  $\mathcal{I} : [1.. \tau] \rightarrow \mathbf{Inp}_M$  where  $\tau \in \mathbb{N} \cup \{\infty\}$ .*

The term “until  $\tau$ ” is written in parentheses to denote that we will omit it, whenever the upper limit is irrelevant for a given aspect. In addition it is clear that an input stream for  $M$  until  $\tau$  will determine an input stream for  $\tau'$  too, if  $1 \leq \tau' \leq \tau$ . For easier usage, we will use  $\mathcal{I}^t$  to denote that  $\mathcal{I}(t)$  for any input stream  $\mathcal{I}$  and  $t \in [1.. \tau]$ <sup>26</sup>. In addition we will use a subscript index to refer on the input function for a single input language  $IL_i$ , such that  $\mathcal{I}_i : [1.. \tau] \mapsto 2^{IL_i}$  which is fully determined by  $\mathcal{I}$ . Note that  $\mathcal{I}_i$  encapsulates input from one input language for every time instant, while  $\mathcal{I}^t$  is encapsulating data for every input language of  $M$  from one time instant. That implies that the input stream contains information for every input language at every time instant. It is required to have this kind of synchronicity of the external information sources, as the reasoning with bridge rules is defined over different inputs and combines them with positive and negative literals. Due to that reliance on awareness between existence and non-existence of information from one or more inputs, we need a way to model that an input source did not provide data too. A simple way would be to set  $\mathcal{I}_i^t$  to the empty set if  $IL_i$  has not provided any data at time instant  $t$ . We will stick to that solution, though it might be necessary

---

<sup>26</sup>Further on we will use the subscript to refer to logical time points for a uniform presentation and notation

to introduce special symbols or some artificial input to represent no answer<sup>27</sup>, opposed to the empty set which might be some kind of answer or information too.

To define the semantics of a reactive Multi-Context System for a given initial configuration of knowledge bases and an input stream for the system we will use the notion of *equilibria streams*.

**Definition 5.2.16** (Equilibria Stream). *Let  $M = \langle C, IL, BR \rangle$  be an reactive Multi-Context System,  $KB$  a configuration of knowledge bases for  $M$ , and  $\mathcal{I}$  an input stream for  $M$  until  $\tau$  where  $\tau \in \mathbb{N} \cup \{\infty\}$ . Then, an equilibria stream of  $M$  given  $KB$  and  $\mathcal{I}$  is a function  $\mathcal{B} : [1..\tau] \rightarrow \text{Bel}_M$  such that*

- $\mathcal{B}^t$  is an equilibrium of  $M$  given  $\mathcal{KB}^t$  and  $\mathcal{I}^t$ , where  $\mathcal{KB}^t$  is inductively defined as
  - $\mathcal{KB}^1 = KB$
  - $\mathcal{KB}^{t+1} = \text{upd}_M(\mathcal{KB}^t, \mathcal{I}^t, \mathcal{B}^t)$ .

In a dual manner, we will refer to the function  $\mathcal{KB} : [1..\tau] \rightarrow \text{Con}_M$  as the configurations stream of  $M$  given  $KB$ ,  $\mathcal{I}$ , and  $\mathcal{B}$ .

The limit  $\tau$  of the input stream is the same limit as for the equilibria stream. In addition if  $\tau \neq \infty$  we will extend the configurations stream by one element, such that  $\mathcal{KB} : [1..\tau'] \rightarrow \text{Con}_M$  where  $\tau' = \tau + 1$ . This is done to produce some kind of final configuration of knowledge bases, because otherwise the last input in the input stream would trigger the computation of an equilibrium, but would not apply the next operators. We call this last configuration in this finite configurations stream *result configuration of  $M$  given  $KB$ ,  $\mathcal{I}$ , and  $\mathcal{B}$* . Based on the definition it is easy to observe that an equilibria stream  $\mathcal{B}$  of  $M$  given  $KB$  and  $\mathcal{I}$  with size  $\tau$  will imply that a substream  $\mathcal{B}'$  of size  $\tau'$ , with  $\tau' \leq \tau$  is an equilibria stream of  $M$  given  $KB$  and  $\mathcal{I}'$ , such that  $\mathcal{I}'$  is the substream of  $\mathcal{I}$  of size  $\tau'$ . Note that an equilibria stream does identify one solution and the configurations stream can be deduced distinctly.

**Example 5.2.17.** *In Example 5.2.13 we have shown how the equilibrium is computed for  $M_{\text{exa}}$ . We will use the same  $M_{\text{exa}}$ , as well as the initial configuration of knowledge bases  $KB = \langle kb_{st} \rangle = \langle \emptyset \rangle$ . Now assume that we have an input stream  $\mathcal{I}$  of size 3, such that  $\mathcal{I}^1 = \langle \{\text{switch}\} \rangle$ ,  $\mathcal{I}^2 = \langle \emptyset \rangle$ , and  $\mathcal{I}^3 = \langle \{\text{switch}\} \rangle$ . In words, we will push the power button of the stove in the first and third time instance. We will now compute the equilibria stream  $\mathcal{B}$  of  $M_{\text{exa}}$ , given  $KB$  and  $\mathcal{I}$ . The input  $l$  is equal to the one given in  $\mathcal{I}^1$ , therefore*

<sup>27</sup>These adaptations can be made by adapting the input languages and would not change the semantics further.

$\mathcal{B}^1 = \mathbf{B}$  from Example 5.2.13. Therefore the new configuration for time instant 2 is computed as

$$\mathcal{KB}^2 = \mathbf{upd}_{M_{exa}}(\mathcal{KB}^1, \mathcal{I}^1, \mathcal{B}^1) = \langle \mathbf{mng}_{st}(\mathbf{app}_{st}^{next}(\mathcal{I}_{st}^1, \mathcal{B}_{st}^1), \mathcal{KB}_{st}^1) \rangle = \langle \{pw\} \rangle.$$

That easy example shows that the intuition of the **next**-operator indeed works as described before. It can be seen how the power state of the stove changes from one time instant to the other, although the reasoning about the power state relates information about the state itself.

$t$	$\mathcal{KB}^t$	$\mathcal{I}^t$	$\mathcal{B}^t$	$\mathbf{app}_{st}^{next}(\mathcal{I}^t, \mathcal{B}^t)$
1	$\langle \emptyset \rangle$	$\langle \{\text{switch}\} \rangle$	$\langle \emptyset \rangle$	$\{\text{setPower}(on)\}$
2	$\langle \{pw\} \rangle$	$\langle \emptyset \rangle$	$\langle \{pw\} \rangle$	$\emptyset$
3	$\langle \{pw\} \rangle$	$\langle \{\text{switch}\} \rangle$	$\langle \{pw\} \rangle$	$\{\text{setPower}(off)\}$
4	$\langle \emptyset \rangle$	–	–	–

Figure 5.1: Streams and applicable operations for  $M_{exa}$

Figure 5.1 is presenting the configurations of knowledge bases, input stream, equilibria stream and the set of applicable next operators for each time instant in a tabular form. Note that the final configuration would be at time point 4. That is a good example why the computation of a final configuration is important. Without the application of the **next**-operator to the knowledge base, the last button push to turn off the stove would have been used for reasoning, but would not been realised in the configuration of knowledge bases.

### 5.2.3 Modelling with Reactive Multi-Context Systems

With the definition of syntax and semantics for a reactive Multi-Context System we now have good understanding how to present some knowledge and reaction to outside influences in a formal way. This section will focus on how to model different important use cases. In addition we will present a bigger example based on the assisted living scenario to give an intuitively accessible insight on our idea of modelling for reactive Multi-Context Systems.

#### Rule Schemata

Till now we only had a look at bridge rules, which used literals built upon elements from the knowledge bases and input languages. Compared to other formalisms (like answer set programming) that means we are only using ground terms. For convenience, and usability in bigger scale, it is imperative for a system like a reactive Multi-Context System to allow the use of variables. Again, we are aiming for an abstract and easily adaptable approach. That is where *rule schemata* are becoming handy. Intuitively, we use a parametrised bridge rule where every instance of the parameters is a bridge rule as presented beforehand.

To have more control over the set of represented bridge rule instances, we want to use conditions to constrain that set. These constraints and conditions will be used to allow concepts like arithmetic operations and comparison relations.

**Definition 5.2.18** (Rule Schemata). *Given a tuple of input languages  $\mathbb{L} = \langle IL_1, \dots, IL_m \rangle$  and the tuple of contexts  $\mathbb{C} = \langle C_1, \dots, C_n \rangle$ . Let  $\mathcal{A}$  be the alphabet of symbols occurring in all possible bridge rules for all contexts  $C_i \in \{C_1, \dots, C_n\}$  over  $\mathbb{C}$  and  $\mathbb{L}$ , the set of symbols  $\mathcal{P}$  be a set of parameters, such that  $\mathcal{A} \cap \mathcal{P} = \emptyset$ .*

- An instantiation term is a string built upon the alphabet  $\mathcal{A}$  for  $\mathbb{C}$  and  $\mathbb{L}$ , and
- an instantiation condition for  $\mathbb{C}$  and  $\mathbb{L}$  is a predicate  $ic(T_1, \dots, T_o)$ , where  $T_1$  to  $T_o$  are strings constructed over the alphabet  $\mathcal{A} \cup \mathcal{P}$ .

A rule schemata  $R$  for  $\mathbb{C}$  and  $\mathbb{L}$  with parameters  $\mathcal{P}$  is of the form

$$H \leftarrow A_1, \dots, A_p, \mathbf{not} A_{p+1}, \dots, \mathbf{not} A_q, D_1, \dots, D_r \quad (5.2)$$

such that  $H, A_1, \dots, A_q$  are strings over  $\mathcal{A} \cup \mathcal{P}$  and  $D_j$  is an instantiation condition for each  $j \in \{1, \dots, r\}$ .

Intuitively we have now a definition on how to write and construct a rule schemata. In Definition 5.2.19 we are now describing the semantics to ensure that only these rules are instantiated which can be constructed by a uniform transformation.

**Definition 5.2.19** (Rule Instantiation). *A bridge rule*

$$r = op \leftarrow a_1, \dots, a_j, \mathbf{not} a_{j+1}, \dots, \mathbf{not} a_k$$

for  $C_i$  over  $\mathbb{C} = \langle C_1, \dots, C_n \rangle$  and  $\mathbb{L} = \langle IL_1, \dots, IL_m \rangle$  is called an instance of a rule schemata  $R$  of the form (5.2) for  $\mathbb{C}$  and  $\mathbb{L}$  with parameters  $\mathcal{P}$  if  $r = (H \leftarrow A_1, \dots, A_p, \mathbf{not} A_{p+1}, \dots, \mathbf{not} A_q)\sigma$  holds for a uniform substitution  $\sigma$ , such that  $\sigma$  substitutes every parameter with its instantiation terms, and for each  $D_j = ic(T_1, \dots, T_o)$ ,  $j \in \{1, \dots, r\}$ , the predicate  $ic(T_1\sigma, \dots, T_o\sigma)$  holds.

For better readability we will utilise the same convention which is used for logic programming, such that parameters will start with uppercase letters. Another convenience is to utilise infix notation for comparisons and arithmetic functions for the instantiation conditions.

**Example 5.2.20.** *In Example 5.2.3 we have introduced the idea about using qualitative values for the temperatures of the stove. For a complete example we will assume that there is a thermostat which will just report the temperature in*

degree Celsius at each time step. We will now illustrate how this could be done by using rule schemata:

$$\begin{aligned} \text{setTemp(cold)} &\leftarrow \text{thermostat}::X, 0 < X < 42 \\ \text{setTemp(hot)} &\leftarrow \text{thermostat}::X, 150 \geq X \geq 42 \end{aligned}$$

These rules express that the operation to set the temperature to be *cold* if the measured temperature is below 42 and above 0 degree, while the temperature is considered to be *hot* if the stove has 42 degree or more (up to a limit of 150 degree).

Note that we assume that the instantiation is done in a smart way. That means if we substitute  $X$  in Example 5.2.20, it will only be substituted by values which will evaluate the instantiation condition to true. Therefore in the first rule, there are only instances where  $X$  is substituted by 1 to 41, and in the second one with 42 to 150 respectively. In addition it might be usable to substitute the rules on demand, based on the given input stream, because the input stream is static during one time instant.

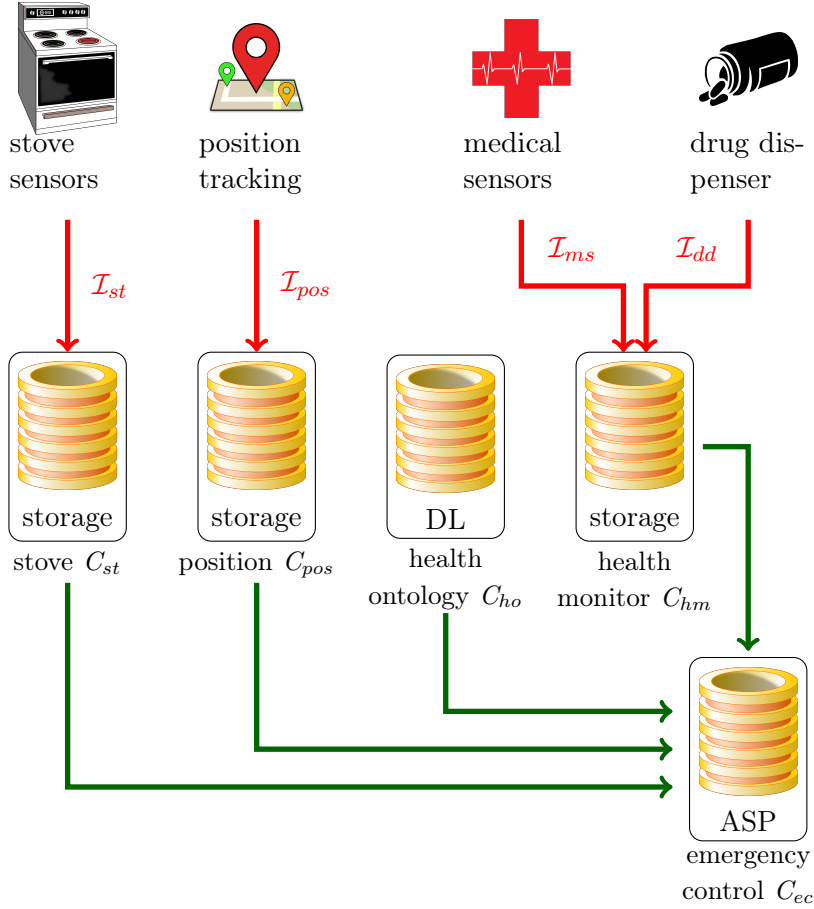
### Example Scenario

Our examples so far have been tailored to show the basic concepts and ideas of the different parts of reactive Multi-Context Systems. To show how the whole system might be used, considering everything we have discussed and defined so far, we will present an extensive example settled in the already illustrated assisted living scenario. It will show how to weave contexts and sensor information from outside sources together into one framework for knowledge integration and dynamic reasoning over time.

Figure 5.2 illustrates the basic environment and configuration of our reactive Multi-Context System  $M_{al}$  to model the reasoning over the patient living in the flat. We have four different providers of outside data, which monitor different things in the daily life of John. That is represented by four input languages  $\mathbb{L} = \langle IL_{st}, IL_{pos}, IL_{hs}, IL_{dd} \rangle$ . Note that we have already shown some parts related to the stove during the examples in Sections 5.2.2 and 5.2.1. To present the whole picture we will refine these given definitions and reiterate them here. Our goal with this example is to model a heterogeneous system, which will do dynamic reasoning over John, a patient suffering dementia. We will model the system to avoid forgetting the stove, as well as a reasoning system which takes the medication and vital signs of John into account to support the patient without bringing him into harms way. In the following we will now present the different parts of the reactive Multi-Context System  $M_{al}$ .

**Sensors and their Input Language:** The stove sensors are still reporting if the power button is pressed. In addition it has thermostat readings for the



Figure 5.2: Structure of  $M_{al}$ 

temperature of the working field. This is represented by the input language

$$IL_{st} = \{\text{switch}\} \cup \{\text{tmp}(T) \mid T \in \mathbb{N}\}.$$

The position tracking can be seen in a very abstract way. It could be some camera based approach, or just a wrist band to pinpoint the position of John. We do not want to go into further detail for how the position is acquired, but we will consider that some measurement is taken to identify the current whereabouts of John. In our example we will get a notification when the supported person enters a room. That leads to the following input language:

$$IL_{pos} = \{\text{enters}(\text{kitchen}), \text{enters}(\text{bathroom}), \text{enters}(\text{bedroom})\}$$

Medical sensors attached to John will report about his current state. Again we will simplify the complex nature of such a dedicated and clinically important sensor device, to get abstract, but useful information provided. The sensors

are able to read the current blood pressure of the patient. Additionally it is possible to find out whether he is asleep or not.

$$IL_{ms} = \{\text{asleep}\} \cup \{\text{bpReading}(R) \mid R \in (\mathbb{N}, \mathbb{N})\}$$

Note that blood pressure is measured for the systolic and diastolic pressure, which is normally written by two natural numbers separated by a slash (e.g. 120/70). For easier readability we will stick to that notation when utilising these values further on. Finally we have the drug dispenser. This device is some automated tool to give John the right amount of medication. We keep it simple and assume that the doses can be retrieved and are consumed directly. When that happens, the sensor will report that some drug has been provided to John. Again, for simplicity we will not delve deeper into the medical topics where the amount would be important too. Therefore the input language for this sensor is as following<sup>28</sup>:

$$IL_{dd} = \{\text{dispensed}(drugA)\}$$

**Contexts with their Bridge Rules** To get a better understanding on how the bridge rules intertwine with the context definitions, we will present them together, although in theory they cannot be defined while all contexts are not settled down already. Note that we will use rule schemata in the same style as bridge rules, to shorten the amount of needed rules. In the following we will consider the following reactive Multi-Context System:

$$M_{al} = \langle\langle C_{st}, C_{pos}, C_{ho}, C_{hm}, C_{ec} \rangle, \\ \langle IL_{st}, IL_{pos}, IL_{ms}, IL_{dd} \rangle, \\ \langle BR_{st}, BR_{pos}, BR_{ho}, BR_{hm}, BR_{ec} \rangle\rangle$$

The context for the stove  $C_{st} = \langle L_{st}, OP_{st}, \mathbf{mng}_{st} \rangle$  has not changed from the introductory examples for reactive Multi-Context System. To give the reasoning of the context more depth, we will assume that the power control of the stove is indeed controlled by the knowledge base of the context. In other words, only if  $pw \in kb_{st}$  holds, the stove will be active. As the used logic, we will still utilise the storage logic  $E_{st} = \{pw, tm(cold), tm(hot)\}$  and the allowed operators are

$$OP_{st} = \{\text{setPower}(off), \text{setPower}(on), \text{setTemp}(cold), \text{setTemp}(hot)\}.$$

The management function has not changed either, such that it is defined as

$$\mathbf{mng}_{st}(OP', kb) = \{pw \mid \text{setPower}(on) \in OP' \vee \\ (pw \in kb \wedge \text{setPower}(off) \notin OP')\} \cup \\ \{tm(t) \mid \text{setTemp}(t) \in OP'\}.$$

---

<sup>28</sup>To show the mechanics it is enough to just assume one drug being used. It is not hard to extend the example.

The bridge rules will first define the behaviour with the power switch and afterwards we will have the quantification of the current temperature. In addition we add some further functionality, which allows the emergency control context to initiate a power shutdown of the stove if it believes that it should be turned off. Therefore  $BR_{st}$  will be set as follows:

$$\begin{aligned} \mathbf{next}(\text{setPower}(on)) &\leftarrow st::\text{switch}, \mathbf{not} \ st:pw, \\ \mathbf{next}(\text{setPower}(off)) &\leftarrow st::\text{switch}, st:pw, \\ \mathbf{next}(\text{setPower}(off)) &\leftarrow ec:\text{turnOff}(stove), \\ \text{setTemp}(cold) &\leftarrow st::\text{tmp}(T), T \leq 42, \text{ and} \\ \text{setTemp}(hot) &\leftarrow st::\text{tmp}(T), 42 < T. \end{aligned}$$

The position tracker context  $C_{pos} = \langle L_{pos}, OP_{pos}, \mathbf{mng}_{pos} \rangle$  will monitor the currently reported position. This straightforward task will be settled by another storage logic  $E_{pos} = \{\text{pos}(P) \mid P \in \{\text{kitchen}, \text{bathroom}, \text{bedroom}\}\}$ . Its allowed operators are to set a new position whenever it gets reported anew, such that  $OP_{pos} = \{\text{setPos}(P) \mid P \in \{\text{kitchen}, \text{bathroom}, \text{bedroom}\}\}$ . To represent that behaviour we will keep old information as long as there is no new information about the current position of John.

$$\mathbf{mng}_{pos}(OP', kb) = \{\text{pos}(P) \mid \text{setPos}(P) \in OP'\} \cup \{k \mid k \in kb \wedge OP' = \emptyset\}$$

For the bridge rules, we want that for every newly reported position the rules will assume that this new position is considered the current position. As this might be interesting for other contexts, we want to have this information during the computation of the equilibrium, as well in the updated knowledge base for the next time instant. Therefore we will have the following two rule schemata for  $BR_{pos}$ :

$$\begin{aligned} \mathbf{next}(\text{setPos}(P)) &\leftarrow pos::\text{enters}(P), \\ \text{setPos}(P) &\leftarrow pos::\text{enters}(P) \end{aligned}$$

Note that we use the same rule twice. Once without the **next**-operator and once with it. That represents that the information should be utilised during the equilibria computation and then it is important to keep that information in the knowledge base too. Intuitively that is important to avoid reasoning on old information, like some reasoning about John not being in the kitchen, although the sensor reported already that he moved on during the current time instant.

The health monitor  $C_{hm} = \langle L_{hm}, OP_{hm}, \mathbf{mng}_{hm} \rangle$  will collect all the important information about John and stores it for further reasoning. Again we will consider this as a simple storage logic which is instantiated by  $E_{hm} = \{\text{status}(asleep), m(\text{drug}A), bp(\text{high}), bp(\text{normal})\}$ . The different used predicates are pretty self explanatory. **Status** will signal if John is asleep, **m** stands for active medication and tracks if some drugs should be in John's blood

cycle, and **bp** has a quantified value for the blood pressure. Note that getting a reading of the blood pressure cannot be done all the time, therefore it is imperative to store this information. In addition it is important to keep track of the current medication, as this is only reported once by the drug dispenser. A total contrast to that is the sleeping information, as the medical sensor will provide that data continuously while John is sleeping. The allowed operations for the context are to set the different readings, such that  $OP_{hm} = \{\text{setStatus}(asleep), \text{setBP}(high), \text{setBP}(normal), \text{setMed}(m(drugA))\}$ . The management function will capture the different properties of the operators. Blood pressure has to be either high or low, asleep will need to be added every time anew, and the medication needs to be added only for our example.

$$\begin{aligned} \mathbf{mng}_{hm}(OP', kb) = & \\ & \{\text{status}(asleep) \mid \text{setStatus}(asleep) \in OP' \vee \text{status}(asleep) \in kb\} \cup \\ & \{\text{bp}(high) \mid \text{setBP}(high) \in OP' \vee \text{bp}(high) \in kb\} \cup \\ & \{\text{bp}(normal) \mid (\text{setBP}(normal) \in OP' \vee \text{bp}(normal) \in kb) \wedge \\ & \quad \text{setBP}(high) \notin OP'\} \cup \\ & \{m(drugA) \mid \text{setMed}(m(drugA)) \vee m(drugA) \in kb\} \end{aligned}$$

Note how we have defined that high blood pressure has priority over normal blood pressure information in case both might be triggered in some case. An additional remark should be given to the management function on the **status**. We have chosen to use keep the information in the knowledge base by the operator use, to show that our desired behaviour can be achieved by the use of bridge rules too.  $BR_{hm}$  are defined as follows:

$$\begin{aligned} \text{setStatus}(asleep) &\leftarrow hs::asleep \\ \text{next}(\text{setBP}(high)) &\leftarrow hs::bpReading(R), R \geq 140/90 \\ \text{next}(\text{setBP}(normal)) &\leftarrow hs::bpReading(R), R < 140/90 \\ \text{next}(\text{setMed}(m(drugA))) &\leftarrow dd::dispensed(drugA) \end{aligned}$$

The status of being asleep is just used for reasoning during the current reasoning, while all other information will be incorporated in the next knowledge base. Note that this means that the medication as well as the blood pressure reading will not be taken into consideration for the computation of the current equilibrium, but needs to be considered in later equilibria due to the fact that this information is directly added for the next time instant.

To have reasoning about the correlation of medical readings and medication, we will utilise a health ontology  $C_{ho} = \langle L_{ho}, OP_{ho}, \mathbf{mng}_{ho} \rangle$ . Due to the expert system like touch, which will only give information, this context does not take any information from the outside world. In addition it does not need to have any considerations about the current belief state of the other contexts. Therefore it has no operators (such that  $OP_{ho} = \emptyset$ ) and the management function is just

the identity (i.e.  $\mathbf{mng}_{ho}(OP', kb) = kb$ ). The used logic will be a description logic, which will be implemented as shown in Example 5.2.1. We will use a very simplified ontology, to show how such a system might be consolidated<sup>29</sup>. The description logic will consist of the following two rules:

$$\begin{aligned} drugA &\sqsubseteq \exists contains.ephedrine \\ \exists contains.ephedrine &\sqsubseteq causesHighBP \end{aligned}$$

In words, this ontology says that **drugA** contains ephedrine. For medical applications it is important to know that this substance might cause high blood pressure.

The last context, which is the emergency control  $C_{ec} = \langle L_{ec}, OP_{ec}, \mathbf{mng}_{ec} \rangle$ , will be responsible to detect emergencies in the environment. The reasoning system is an answer set program, so the logic is used as shown in Example 5.1.8. We will use one single operator  $OP_{ec} = \{\mathbf{add}(R)\}$ , which will allow to add any string  $R$  to the answer set program. It is considered that  $R$  should only consist of answer set rules. Each rule is added by the management function, such that  $\mathbf{mng}_{ec}(OP', kb) = kb \cup \{R \mid \mathbf{add}(R) \in OP'\}$ . Because the other contexts are already incorporating and quantifying the information from the input stream, the bridge rules of this context will only utilise beliefs from other contexts. This will illustrate how the system can query other formalisms to get to its own conclusions. The bridge rules are as follows:

$$\begin{aligned} \mathbf{add}(\text{oven}(on, hot).) &\leftarrow st:pw, st:tm(hot) \\ \mathbf{add}(\text{humanPos}(P).) &\leftarrow pos:pos(P) \\ \mathbf{add}(\text{status}(asleep).) &\leftarrow hm:asleep \\ \mathbf{add}(\text{highBP}.) &\leftarrow hm:bp(high) \\ \mathbf{add}(\text{highBPMed}.) &\leftarrow ho:causesHighBP(D), hm:m(D) \end{aligned}$$

Note that we use the oven information from the first rule only to have the information in case there might be an emergency (e.g. the oven is on and hot, while no person is in the kitchen). In the second rule use rule schemata to keep track of the position of the human (i.e. John) in every case. As the initial configuration for that context, we will use the answer set program  $kb_{ec}$  as follows:

$$\begin{aligned} \mathbf{alert}(stove) &\leftarrow \text{oven}(on, hot), \mathbf{not} \text{humanPos}(kitchen), \mathbf{not} \text{status}(asleep). \\ \mathbf{turnOff}(stove) &\leftarrow \text{oven}(on, hot), \text{status}(asleep). \\ \mathbf{call}(medAssist) &\leftarrow \text{highBP}, \mathbf{not} \text{highBPMed}. \end{aligned}$$

<sup>29</sup>We want to show that such a DL reasoner can be used, of course a real ontology would be a massive pool of knowledge and we would like to manipulate the A-Box with the management functions and operators. For the sake of simplicity and easily graspable examples we will stick to this very simplistic representation.

In words, that program will be controlling if some emergency is occurring. First, it will come to the conclusion that some alarm should be sounded, if the oven is turned on and John is awake but not in the kitchen. Because John should not be bothered if he falls asleep, the second rule will derive that the stove should be turned off if the stove is turned on, but John sleeps. Note that this predicate `turnOff`, is the same which is used for the second power-off bridge rule of the stove context. So if the answer set program concludes that the stove should be turned off, it will be done by the interplay of the bridge rules. The last rule will call a medical assistant in the case that high blood pressure is detected, but no good explanation (i.e. a medication that causes it) is present.

**Example of an Equilibria Stream** Now with all set up, we will present on how this entire reactive Multi-Context System will react to a given input stream of seven different time instants. Figure 5.3 shows how the different contexts change over time. We do omit from  $\mathcal{KB}^t$  and  $\mathcal{B}^t$  all fixed information, such as the answer set program encoding for  $C_{ec}$ , to make it easier to read and compute. Our example starts with John at time  $t = 1$  being in the kitchen and he turns the stove on. This effect is then visible at  $t = 2$ , together with an increase in the temperature of the stove. In addition he gets his medication from the drug dispenser. Because the blood pressure reading is not done in every time instant, we see that this time instant is one of these occurrences where this is the case. After getting his drugs, John goes into his bedroom at time  $t = 3$ . The information is stored via a next room, so his position change is reflected in the configuration too. Here his blood pressure is measured again, which is still normal. John might have forgotten the stove already or maybe he waits till it is hot enough to cook something. Now at time instant  $t = 4$  a high blood pressure reading occurs. Because the bridge rules for this reading are **next**-rules, the equilibrium is not reflecting that change. We do see that in the next time instant the pressure is changed to be recognised as being high. During  $t = 5$  the system does reasoning about the high blood pressure. It is very likely to be caused by the recently taken pills, therefore no medical assistant is called here. At  $t = 6$  the oven got heated up and is hot now. Alas, John fell asleep too and should not be woken up again. Therefore the emergency context decides to turn the stove off. Because we are looking at a finite substream, we will look at  $t = 7$  as the result of that input stream, where we can see that the command from the emergency control has been executed by the stove context.

### Modelling Aspects of Reactive Multi-Context Systems

With the running example one can see how natural it can deal with knowledge integration of different formalisms and dynamic environments. Now we want to focus on different aspects for which we will discuss how to model them. This discussion will eventually lead into generic modelling techniques for reactive

$t$	$\mathcal{KB}^t$	$\mathcal{I}^t$	$\text{app}_i^{\text{now}}(\mathcal{I}^t, \mathcal{B}^t)$	$\mathcal{B}^t$	$\text{app}_i^{\text{next}}(\mathcal{I}^t, \mathcal{B}^t)$
1	$\langle \emptyset, \{\text{pos}(\text{kitchen})\}, \{\text{bp}(\text{normal})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(19), \text{switch}\}, \emptyset, \{\text{bpReading}(135/86)\}, \emptyset \rangle$	$\langle \{\text{setTemp}(\text{cold})\}, \emptyset, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{kitchen}))\} \rangle$	$\langle \{\text{tm}(\text{cold})\}, \{\text{pos}(\text{kitchen})\}, \{\text{bp}(\text{normal})\}, \emptyset, \{\text{humanPos}(\text{kitchen})\} \rangle$	$\langle \{\text{setPower}(\text{on})\}, \emptyset, \{\text{setBP}(\text{normal})\}, \emptyset, \emptyset \rangle$
2	$\langle \{\text{pw}\}, \{\text{pos}(\text{kitchen})\}, \{\text{bp}(\text{normal})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(21)\}, \emptyset, \emptyset, \{\text{dispensed}(\text{drugA})\} \rangle$	$\langle \{\text{setTemp}(\text{cold})\}, \emptyset, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{kitchen}))\} \rangle$	$\langle \{\text{tm}(\text{cold}), \text{pw}\}, \{\text{pos}(\text{kitchen})\}, \{\text{bp}(\text{normal})\}, \emptyset, \{\text{humanPos}(\text{kitchen})\} \rangle$	$\langle \emptyset, \emptyset, \{\text{setMed}(\text{m}(\text{drugA}))\}, \emptyset, \emptyset \rangle$
3	$\langle \{\text{pw}\}, \{\text{pos}(\text{kitchen})\}, \{\text{bp}(\text{normal}), \text{m}(\text{drugA})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(27)\}, \{\text{enters}(\text{bedroom})\}, \{\text{bpReading}(138/89)\}, \emptyset \rangle$	$\langle \{\text{setTemp}(\text{cold})\}, \{\text{setPos}(\text{bedroom})\}, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{bedroom})), \text{add}(\text{highBPMed})\} \rangle$	$\langle \{\text{tm}(\text{cold}), \text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{normal})\}, \emptyset, \{\text{humanPos}(\text{bedroom}), \text{highBPMed}\} \rangle$	$\langle \emptyset, \{\text{setPos}(\text{bedroom})\}, \{\text{setBP}(\text{normal})\}, \emptyset, \emptyset \rangle$
4	$\langle \{\text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{normal}), \text{m}(\text{drugA})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(36)\}, \emptyset, \emptyset, \{\text{bpReading}(148/97)\}, \emptyset \rangle$	$\langle \{\text{setTemp}(\text{cold})\}, \emptyset, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{bedroom})), \text{add}(\text{highBPMed})\} \rangle$	$\langle \{\text{tm}(\text{cold}), \text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{normal})\}, \emptyset, \{\text{humanPos}(\text{bedroom}), \text{highBPMed}\} \rangle$	$\langle \emptyset, \emptyset, \{\text{setBP}(\text{high})\}, \emptyset, \emptyset \rangle$
5	$\langle \{\text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{high}), \text{m}(\text{drugA})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(42)\}, \emptyset, \emptyset, \{\text{bpReading}(146/95)\}, \emptyset \rangle$	$\langle \{\text{setTemp}(\text{cold})\}, \emptyset, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{bedroom})), \text{add}(\text{highBPMed}), \text{add}(\text{highBP})\} \rangle$	$\langle \{\text{tm}(\text{cold}), \text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{high})\}, \emptyset, \{\text{humanPos}(\text{bedroom}), \text{highBPMed}, \text{highBP}\} \rangle$	$\langle \emptyset, \emptyset, \{\text{setBP}(\text{high})\}, \emptyset, \emptyset \rangle$
6	$\langle \{\text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{high}), \text{m}(\text{drugA})\}, \emptyset, \emptyset \rangle$	$\langle \{\text{tmp}(51)\}, \emptyset, \emptyset, \{\text{asleep}\}, \emptyset \rangle$	$\langle \{\text{setTemp}(\text{hot})\}, \emptyset, \{\text{setStatus}(\text{asleep})\}, \emptyset, \emptyset, \{\text{add}(\text{humanPos}(\text{bedroom})), \text{add}(\text{highBPMed}), \text{add}(\text{highBP}), \text{add}(\text{oven}(\text{on}, \text{hot})), \text{add}(\text{status}(\text{asleep}))\} \rangle$	$\langle \{\text{tm}(\text{hot}), \text{pw}\}, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{high})\}, \emptyset, \{\text{humanPos}(\text{bedroom}), \text{highBPMed}, \text{highBP}, \text{oven}(\text{on}, \text{hot}), \text{status}(\text{asleep}), \text{turnOff}(\text{stove})\} \rangle$	$\langle \{\text{setPower}(\text{off})\}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$
7	$\langle \emptyset, \{\text{pos}(\text{bedroom})\}, \{\text{bp}(\text{high}), \text{m}(\text{drugA})\}, \emptyset, \emptyset \rangle$				

Figure 5.3: Streams and applicable operations for the example scenario given by  $M_{al}$

Multi-Context Systems which should be very helpful in the typical application area. Of course, for specific needs these approaches may still need refinements and fine tuning, but these would be very domain specific.

We will discuss how to incorporate stream data in general. Afterwards the difference between operational (i.e. **next**-operator rules) and declarative (i.e. the other operator rules) bridge rules with their different intended uses will be focused on. Then we will discuss the usage of time, similar to the discussions already done for original reactive Multi-Context Systems in Section 5.1, followed by a concept to handle inconsistent stream data. In the end a discussion on forgetting and data retention will be given. Note that we have already used many of these concepts in the previous example and we will refer to already presented rules as examples for the different methods, where applicable. In addition we will give hints and ideas and we may omit the whole definition of additional operators and tweaks to the management function if it would make the example too unclear and unfocused.

**Incorporating Stream Data** The main idea of Multi-Context Systems has been to see bridge rules as a way to translate information and knowledge from one context to another. For reactive Multi-Context Systems we have used this view on bridge rules and extended the concept to allow that translation for external sources, represented by input languages and their respective input streams. One such bridge rule might just incorporate and translate the information from an input stream. Moreover it allows one to pass information to the context in other ways than it is represented in the body. This enables us to focus on different aspects of the given information, such as relevant parts of the information, adding further meta-information, execute arithmetic operations on the data, and much more. In the previously shown example we had many different input streams. Especially Context  $C_{st}$  utilised this abstraction and focus on important information in the rules about the temperature:

$$\begin{aligned} \text{setTemp}(\text{cold}) &\leftarrow st::\text{tmp}(T), T \leq 42, \text{ and} \\ \text{setTemp}(\text{hot}) &\leftarrow st::\text{tmp}(T), 42 < T. \end{aligned}$$

In that specific case the rules only translated the relevant information for the context, namely whether the stove is hot or not. Therefore we are only focusing on the relevant part, while we are using meta-information too (i.e. when does something start to be hot). In contrast to that we could use some slightly different rule to store specific values in some cases:

$$\text{addTemp}(T) \leftarrow st::\text{tmp}(T), 42 < T$$

This rule would just ignore the temperature of the stove if it is cold and report the current temperature in degree Celsius to the context otherwise.



**Operational and Declarative Bridge Rules** The main goal of this part is to give a better understanding on the difference between **next**-operator rules (i.e. operational rules) and those which are not **next**-operator rules (i.e. declarative rules). Very briefly said, the declarative bridge rules utilise the provided information in order to decide which set of belief sets are equilibria, while operational bridge rules are only operated on afterwards to update the knowledge base itself. Considering the switch behaviour of our example, we had this switch for the stove which acted as an impenetrable truth which cannot be discussed by the contexts. To model this change of truth we used the operational bridge rules

$$\begin{aligned} \mathbf{next}(\text{setPower}(on)) &\leftarrow st::\text{switch}, \mathbf{not} \ st:pw, \text{ and} \\ \mathbf{next}(\text{setPower}(off)) &\leftarrow st::\text{switch}, st:pw \end{aligned}$$

to model this switch logic. In this case the value gets flipped every time the switch is used during a time instant. Note that such a flip cannot be done for declarative bridge rules, as discussed in Section 5.1<sup>30</sup>. Lets assume we need to incorporate the knowledge about the fact that the switch has been used, then some rule like

$$\text{switchpressed} \leftarrow st::\text{switch}$$

can be used to express this. Of course, in some cases it is helpful and desired to use the same bridge rule in its operational and declarative version, as it has been shown for Context  $C_{pos}$ . There the current position is updated for the equilibria computation by the rule

$$\mathbf{next}(\text{setPos}(P)) \leftarrow pos::\text{enters}(P)$$

and it is stored by

$$\text{setPos}(P) \leftarrow pos::\text{enters}(P)$$

because this information is not provided in a continuous manner.

**Time in Reactive Multi Context Systems** Due to the explicit use of the discrete logical time for the definition of the input stream, the whole reactive Multi-Context System uses these time instants implicitly for the equilibria stream, update functions, and stream of configurations. In the following bridge rule examples, we will consider this logical time of time instants, although it might be necessary to operate on explicit physical time or other kinds of logical time. Both of these additional time concepts can be incorporated on the level of bridge rule modelling. In fact in both cases it is just a matter of incorporating stream data to get this information integrated into the reactive

<sup>30</sup>As a simple rule one can say, that every direct flipping cannot be done by declarative rules, as the flip itself would prevent every equilibrium candidate to be an equilibrium

Multi-Context System. Then the overall handling is the same as the usage of the time instants. To show how to incorporate time into some reasoning, we will consider some monitoring context  $C_{log}$ , which will have a history on the temperatures of the stove.

$$\mathbf{next}(\mathbf{add}(\mathbf{tmpAtTime}(Temp, T))) \leftarrow st::\mathbf{tmp}(Temp), c::\mathbf{now}(T)$$

is an operational bridge rule, which utilises an input stream  $\mathcal{I}_c$ , which acts as a clock. It will add the temperature at the current time to the knowledge base and therefore collects the different temperatures over time. The expression  $\mathbf{tmpAtTime}(Temp, T)$  should be read such that at time  $T$  the temperature  $Temp$  has been measured. Such a logging context, together with the time aspect now allows to do reasoning which includes queries on the past. One example for that might be a query whether the stove has been hot during the last 10 minutes<sup>31</sup>:

$$\begin{aligned} \mathbf{add}(\mathbf{recentlyHot}) &\leftarrow log:\mathbf{tmpAtTime}(Temp, T'), Temp > 42, \\ &c::\mathbf{now}(T), T' \geq T - 10 \end{aligned}$$

If there is no external clock, it is viable and simple to create a clock context, which will keep track of the current time instant. Note that this was one of the more delicate problems discussed for the original reactive Multi-Context Systems. Our Context  $C_{clock}$  will start with an empty knowledge base and might be modelled with a simple storage logic. The following set of bridge rules will be all needed in terms of mechanics to model such a time instant clock:

$$\begin{aligned} \mathbf{setTime}(\mathbf{now}(0)) &\leftarrow \mathbf{not} \text{ clock:timeAvailable} \\ \mathbf{next}(\mathbf{add}(\mathbf{timeAvailable})) &\leftarrow \text{clock:now}(0) \\ \mathbf{next}(\mathbf{setTime}(\mathbf{now}(T + 1))) &\leftarrow \text{clock:now}(T) \end{aligned}$$

Rule one is there to ensure that if the context is not initialised, it will get initialised with the time instant 0 for the first computation of the equilibrium. Then the second rule will set a flag that the context is initialised because it is believing that the current time instant is 0. In the third rule the time instant is advanced by one for each computation of a new configuration for the reactive Multi-Context System.

The elegant thing of this kind of solution is that awareness of time can be modelled by the bridge rules. Therefore the system does not need any mechanics to provide time on its own. That also allows mixing of different time concepts in an independent and general way.

---

<sup>31</sup>Note that we just figuratively call this 10 minutes. It is completely clear that this would either need a specific time concept, like a physical time clock, or some assumptions towards the computation time of the equilibria stream

**Handling Inconsistent Stream Data** When dealing with input streams, all the information provided by them are facts, which cannot be changed or altered any more. After incorporating them into the system, they got translated into the Contexts and start to be debatable point of views. There are many situations where it might happen that sensor data is inconsistent. In the following we will consider one specific example on how to model conflicting sensor data, with specific properties. Assume some set of sensors, where reliability and detail level are inversely proportional (i.e. the most reliable context has the least level of detail).

The reactive Multi-Context System  $M_{iss} = \langle C_{iss}, \mathbf{IL}_{iss}, \mathbf{BR}_{iss} \rangle$  for handling inconsistent sensor streams should collect as much detailed and consistent sensor readings from such a set of sensors as possible.  $\mathbf{IL} = \langle IL_1, \dots, IL_k \rangle$  represent the  $k$  number of sensors, and  $C_{cv} \in C_{iss}$  is a consistency verifying context, which should provide consistent sensor readings to other contexts. First we define a bridge rule for the context to incorporate all of the sensor data of the form

$$\text{addC}((C), (j)) \leftarrow j::D$$

for every  $j \in \{1, \dots, k\}$ . The management function will use the operator  $\text{addC}(C, j)$  to add every sensor reading together with its source into the knowledge base. We now assume that we have a reliability preference relation  $\succ$  over the sensors, such that  $IL_1 \succ \dots \succ IL_k$ , that means that  $IL_1$  has the highest priority in terms of reliability. To have a way of justification of consistency between information, we will consider the property  $\text{cons}(kb)$  which holds if  $kb \in KB_{cv}$  is consistent<sup>32</sup>. Given a set of operations  $OP$ , we define sets of input data from each sensor such that  $\text{inp}'_j = \{d \mid \text{addC}(d, j) \in OP\}$  for  $j \in \{1, \dots, k\}$ . Then we assume for the case where no sensor data is available that  $\text{inp}'_0(OP) = \emptyset$  and let

$$\text{inp}_j(OP) = \begin{cases} \text{inp}'_{j-1}(OP) \cup \text{inp}'_j & \text{if } \text{cons}(\text{inp}'_{j-1}(OP) \cup \text{inp}'_j) \\ \text{inp}'_{j-1}(OP) & \text{otherwise.} \end{cases}$$

Finally the management function then can just add  $\text{inp}'_k$  to the knowledge base of the context to have the set of reliable and precise data integrated. Intuitively the idea is to start with no information and then start to add reliable information. Further on every next iteration will be more and more unreliable and will only be added if the more exact information is still in line with the previously collected information.

Without much change it would be possible to utilise additional meta data, by just changing the semantics of the consistency check. In the case where no preferences are available, this method can be easily adapted. If, for example, one is interested in using a subset-maximal consistent subset of all sensor

<sup>32</sup>This notion of consistency is meant to be with respect to the represented domain of the sensor data, not overall consistency

data, the management function would add this maximal set  $inp_{mx}$  to the knowledge base.  $inp_{mx}$  is a maximal set where  $inp_{mx} \subseteq \{d \mid \text{addC}(d, j) \in OP\}$  for  $j \in \{1, \dots, k\}$  and  $\text{consinp}_{mx}$  holds.

Note that we are dealing with inconsistent stream data in this section, while Section 5.2.4 will provide means to deal with inconsistencies on the formalism level. There it will be discussed on how to avoid situations which will lead to a non-existence of equilibria due to inconsistent belief states.

**Forgetting, Data Retention, and Filtering** During the span of time instants, a reactive Multi-Context System will collect and store a huge diversity of data. That may lead to an unnecessary high number of already deprecated information, which may only slow down the computation of currently important decisions. Therefore it is desirable to have mechanics to forget old knowledge again when it is no longer needed, while keeping notable possibly even older information. Note that by the usage of the operational bridge rules it is easy to remove any information already stored, so the basic problem of removing data is not an issue. The technique of time windows over some span of time is usually used in such scenarios. So we will model a context which will use such windows. We will further show that we can adjust the window size via bridge rules.

Again, we consider the assisted living scenario. We extend the input languages to have  $IL_c$  which provides the current time. In addition we use a further context  $C_{stE}$ , which logs emergency alerts raised by the emergency control  $C_{ec}$ . This context will log the stove alerts with a time stamp. In addition it manages its own time window with the element  $\text{winE}(t)$  as part of its knowledge base. The basic idea is, to log how long the current alert is in place and if the alert is consistently unaddressed over some time it will lead to an emergency situation. This behaviour is represented by the following rule schemata:

$$\begin{aligned} \text{next}(\text{add}(\text{alert}(\text{stove}, T))) &\leftarrow c::\text{now}(T), ec:\text{alert}(\text{stove}). \\ \text{next}(\text{del}(\text{alert}(\text{stove}, T))) &\leftarrow stE:\text{alert}(\text{stove}, T), \text{not } ec:\text{alert}(\text{stove}). \\ \text{add}(\text{emergency}(\text{stove})) &\leftarrow c::\text{now}(T), ec:\text{alert}(\text{stove}), stE:\text{alert}(\text{stove}, T'), \\ &\quad stE:\text{winE}(Y), |T - T'| \geq Y. \end{aligned}$$

In rule one the current alert state, together with the current time, will be added to the knowledge base of the emergency logging context. Then if the alert is resolved and no longer emerging, the second rule will clean up again and remove all the other tracked alerts in the next time instant. The third rule is now checking the alert history. If the currently tracked alert for the stove is for the set window or longer in place, then an emergency state should be deduced. Note that the other contexts now may inform some medical assistant or just turn off the stove as it is done if John is asleep.

We have shown how to use a history and forget information based on the current state of other contexts. Now we want to figure a situation where the window size is changed based on some reasoning. To consider such a case we will stick to the idea of emergency tracking again. We will use a context  $C_{ed}$ , which has the task to detect emergencies. In this example the emergency detection is done in two steps. First some information might hint towards an emergency, that is when an emergency is suspected. Based on this suspicion more information is collected to confirm or retract the emergency, which is the second step. To keep it simple, we will assume an input language  $IL_s$  to represent the sensor data of possible observations. Each of these observations might trigger one or more emergencies  $e_1, \dots, e_m$  to be suspected or confirmed.  $C_{ed}$  is then adjusting the window of one or more families of observations. How the context works internally is not crucial to present the mechanics, but we will assume the following properties:

- $C_{ed}$  can signal that an emergency  $e_i$  is suspected ( $\text{susp}(e_i)$ ) or confirmed ( $\text{conf}(e_i)$ ).
- $C_{ed}$  has information about default and actual window sizes for an observation  $p$  in the form of  $\text{defWin}(p, w)$  and  $\text{win}(p, w)$  respectively.
- $C_{ed}$  has information on the relevant time window, in case an emergency is suspected, represented by  $\text{rel}(p, e, w)$ .

To model the change of window-sizes based on these assumptions can be modelled by the following rule schemata:

$$\begin{aligned} \mathbf{next}(\text{set}(\text{win}(P, X))) &\leftarrow ed:\text{defWin}(P, X), \mathbf{not} ed:\text{susp}(E). \\ \mathbf{next}(\text{set}(\text{win}(P, Y))) &\leftarrow ed:\text{rel}(P, E, Y), ed:\text{susp}(E). \\ \mathbf{alarm}(E) &\leftarrow ed:\text{conf}(E). \end{aligned}$$

Intuitively the rules mean that if no emergency is suspected, the default window is set for the observation. In case some emergency is suspected, then the second rule will set the emergency based window for the observation. Finally the last rule will raise an alarm if the emergency could be confirmed. Note that it might happen that one property is part of different windows. Then it is important to have the management function handle that situation (e.g. only adding the biggest window<sup>33</sup>).

What is missing now, is how to add sensor data and how retention and deletion of different observation is handled. We will present an easy way to do that, such that the information is added by

$$\mathbf{next}(\text{add}(P(T))) \leftarrow c::\text{now}(T), s::P.$$

<sup>33</sup>One might add priorities to the windows too, such that the highest priority value for each observation is used instead

We don't need to do anything to keep information. To remove observations, we will utilise the set window and the current time, such that

$$\mathbf{next}(\text{del}(P(T'))) \leftarrow \text{ed:P}(T'), c::\text{now}(T), \text{ed:win}(P, Z), T' < T - Z.$$

#### 5.2.4 Inconsistency Management

When working with different knowledge representation formalisms, it is an indisputable fact that inconsistencies can occur. Especially in a dynamic setting, where the knowledge bases of the contexts change over time and the input stream contains incontestable facts. If some inconsistency occurs during the computation of the equilibria stream, it is possible that due to that inconsistent state no belief states for the reactive Multi-Context System is an equilibrium. In that case the whole formalism is hindered and rendered totally useless, because it is not possible to go on towards the next time instant without an equilibrium. Therefore no new knowledge bases for the contexts can be processed and the computation ends prematurely. Inconsistency management is the general topic which is dealing with such problems by providing solutions, that either no inconsistencies can occur or the reason of inconsistencies is found and repaired. In other words, the equilibria stream requires the existence of an equilibrium at every time instant.

We need to classify the many reasons of inconsistencies in reactive Multi-Context Systems. In general we can distinguish between *local inconsistencies* and *global inconsistencies*. The former type occurs if one context is inconsistent and the cause for that purely emerges from the embedded context logic. These embedded formalism issues are highly dependent from the used logic and therefore we will not focus on this kind of inconsistencies. On the other hand, the latter type of inconsistency is caused by the mechanics of the reactive Multi-Context System. Reasons for global inconsistencies include

- the absence of a belief state which can be accepted by every context,
- bridge rules whose operations lead to inconsistencies in the context,
- bridge rules which propose changes such that the bridge rule itself is no longer applicable, and
- input streams which force bridge rules to model a flow of information which cannot lead to an equilibrium.

In this section we will focus mainly on the non-existence of equilibria. Related work on managed Multi-Context Systems [Eiter et al., 2014, Weinzierl, 2014] has addressed inconsistency management before and our ideas are based on their approaches toward that topic. They have established two notions, *diagnosis* and *explanations*, where the former focuses on identifying adaptations and alterations of rules to restore consistency, while the latter is dedicated to analyse which combination of rules is responsible for an inconsistency. Our

approach here will be fashioned after the diagnosis notion. We can omit explanations, because it has been shown that the both notions are the dual of each other<sup>34</sup>.

To achieve a common ground where we can focus solely on the global consistency, we will first need some condition for the existence of equilibria. Therefore we define the notion of consistency for a given reactive Multi-Context System.

**Definition 5.2.21** (Consistency). *Let  $M$  be a reactive Multi-Context System,  $\text{KB}$  a configuration of knowledge bases for  $M$ , and  $\mathcal{I}$  an input stream for  $M$ . Then,  $M$  is consistent with respect to  $\text{KB}$  and  $\mathcal{I}$  if there exists an equilibria stream of  $M$  given  $\text{KB}$  and  $\mathcal{I}$ .  $M$  is strongly consistent with respect to  $\text{KB}$  if, for every input stream  $\mathcal{I}$  for  $M$ ,  $M$  is consistent with respect to  $\text{KB}$  and  $\mathcal{I}$ .*

Note that we introduced two versions of global consistency, depending on whether a particular input stream is assumed or every input stream is considered. In addition it is obvious that for a given configuration of knowledge bases a strongly consistent reactive Multi-Context System is consistent for one input stream too, but not vice versa. To verify consistency in general is highly complex, because every possible equilibria stream needs to be verified to ensure that this property holds. Therefore we are introducing sufficient properties of a given reactive Multi-Context System to ensure its consistency, without the need of this by-case verification.. First we need to ensure that the context formalism will be coherent in such a way that its logic accepts every knowledge base in its semantics. We call this property *total coherence*.

**Definition 5.2.22** (Total Coherence). *A context  $C_i$  is totally coherent if  $\text{acc}_i(kb) \neq \emptyset$ , for every  $kb \in \text{KB}_i$ .*

Intuitively, if some context is not totally coherent, it might happen that one knowledge base does not compute into a belief set. That would imply that this context cannot produce a belief set and making it therefore impossible for any belief state to be an equilibrium for that reactive Multi-Context System. Now that we have a notion where such a behaviour is not an issue, we need to take a look at bridge rules. As listed above, bridge rules can have a manifold impact on consistency, therefore we will disallow rule constructs which may interfere with the computation of an equilibrium. It has been part of different previous discussions that self dependent bridge rules might cause problems. So a notion of dependence is needed.

**Definition 5.2.23** (Information Dependence). *Given a reactive Multi-Context System  $M = \langle \langle C_1, \dots, C_n \rangle, \text{IL}, \text{BR} \rangle$ ,  $\triangleleft_M$  is the binary relation over contexts of  $M$  such that  $(C_i, C_j) \in \triangleleft_M$  if there is a bridge rule  $r \in \text{BR}_i$  and  $j:b \in \text{body}(r)$*

---

<sup>34</sup>We invite the interested reader to have a look at the PhD-thesis of Antonius Weinzierl [Weinzierl, 2014] for further information on the duality and on explanations.

for some  $b$ . If  $(C_i, C_j) \in \triangleleft_M$ , also denoted by  $C_i \triangleleft_M C_j$ , we say that  $C_i$  depends on  $C_j$  in  $M$ , dropping the reference to  $M$  whenever unambiguous.

This relation defines that some context is receiving information from some other context and therefore its knowledge base is altered based on this dependence on this data. If this dependence is resulting in a cycle, it might turn out to be such an instance, where the operators are triggered by values they will change afterwards.

**Definition 5.2.24** (Acyclicity). *An reactive Multi-Context System  $M$  is called acyclic if the transitive closure of  $\triangleleft_M$  is irreflexive.*

With acyclic reactive Multi-Context Systems we can now propose that this property, together with total coherence is sufficient to ensure strong consistency.

**Proposition 5.2.25.** *Let  $M = \langle\langle C_1, \dots, C_n \rangle, \text{IL}, \text{BR}\rangle$  be an acyclic reactive Multi-Context System such that every  $C_i$ ,  $1 \leq i \leq n$ , is totally coherent, and  $\text{KB}$  a configuration of knowledge bases for  $M$ . Then,  $M$  is strongly consistent with respect to  $\text{KB}$ .*

*Proof.* Let  $M = \langle\langle C_1, \dots, C_n \rangle, \text{IL}, \langle \text{BR}_1, \dots, \text{BR}_n \rangle\rangle$  be a reactive Multi-Context System that is acyclic with totally coherent contexts. We first prove that  $M$  has an equilibrium given  $\text{KB}$  and  $\text{l}$ , for any knowledge base configuration  $\text{KB} = \langle kb_1, \dots, kb_n \rangle$  for  $M$  and input  $\text{l}$  for  $M$ .

We prove this by induction on the number of contexts of  $M$ , making use of the following simple observation: if  $M$  does not have cycles, then there exists some  $i \in \{1, \dots, n\}$  such that  $\text{ref}_r(j, i)$  does not hold for any  $j \in \{1, \dots, n\}$  and  $r \in \text{BR}_j$ , where  $\text{ref}_r(j, i)$  holds precisely when  $r$  is a bridge rule of context  $C_j$  and  $i:b$  occurs in the body of  $r$ . It is quite easy to see that if this condition is violated then a cycle necessarily exists.

Let  $n = 1$ . Then, since there are no cycles, no bridge rule in  $\text{BR}_1$  contains atoms of the form  $1:b$  in its body. Thus,  $\text{app}_i^{\text{now}}(\text{l}, \text{B})$  does not depend on  $\text{B}$ . Total coherence then immediately implies that  $M$  has an equilibrium given  $\text{KB}$  and  $\text{l}$ .

Let  $n = m + 1$ . We use the above observation, and assume, w.l.o.g., that  $C_1$  is a context for which  $\text{ref}_r(j, 1)$  does not hold for any  $j \in \{1, \dots, m + 1\}$  and  $r \in \text{BR}_j$ . Then, the reactive Multi-Context System

$$M^* = \langle\langle C_2, \dots, C_{m+1} \rangle, \text{IL}, \langle \text{BR}_2, \dots, \text{BR}_{m+1} \rangle\rangle$$

has  $m$  contexts and it is still acyclic. By induction hypothesis, we can conclude that  $M^*$  has an equilibrium given  $\text{KB}^* = \langle kb_2, \dots, kb_{m+1} \rangle$  and  $\text{l}$ . Let  $\text{B}^* = \langle B_2, \dots, B_{m+1} \rangle$  be such equilibrium. Then, since  $C_1$  is assumed to be a totally coherent context, there exists  $B_1 \in \text{BS}_1$  such that  $\text{B} = \langle B_1, B_2, \dots, B_n \rangle$  is an equilibrium of  $M$  given  $\text{KB}$  and  $\text{l}$ . This follows easily from the fact that no set  $\text{app}_i^{\text{now}}(\text{l}, \text{B})$  depends on the choice of  $B_1$ .



We have shown that the existence of an equilibrium for  $M$  is independent of the given knowledge base configuration  $\mathbf{KB}$  for  $M$  and input  $\mathcal{I}$  for  $M$ . This immediately implies that for any input stream  $\mathcal{I}$  for  $M$  (until  $\tau$ ), and any knowledge base configuration  $\mathbf{KB}$  for  $M$ , there exists an equilibria stream of  $M$  given  $\mathbf{KB}$  and  $\mathcal{I}$ .  $\square$

This result is limiting the modelling options for reactive Multi-Context Systems very much. In many cases it is not practical to be that restricting, as cyclicity in bridge rules has proven to provide useful mechanics in previous examples (see Example 5.1.9, 5.1.10 and the example scenario of Section 5.2.3). So it is desirable to allow and utilise cyclic bridge rules and break cycles when they are causing inconsistencies and hindering the computation of an equilibrium. We will do this by introducing a *repair function*  $\mathcal{R}$ , which will remove bridge rules in a local and selective way to recover the broken equilibria stream. To formulate this kind of repair, one needs to remove a set of bridge rules from the reactive Multi-Context System  $M$ . The reactive Multi-Context System  $M'$  which is the same reactive Multi-Context System as  $M$ , but without the set of bridge rules  $R$ , will be denoted as  $M[R]$ .

**Definition 5.2.26** (Repair). *Let  $M = \langle \mathbf{C}, \mathbb{L}, \langle BR_1, \dots, BR_n \rangle \rangle$  be a reactive Multi-Context System,  $\mathbf{KB}$  a configuration of knowledge bases for  $M$ ,  $\mathcal{I}$  an input stream for  $M$  until  $\tau$  where  $\tau \in \mathbb{N} \cup \{\infty\}$ , and  $ABR_M = \bigcup_{1 \leq i \leq n} BR_i$  the set of all bridge rules of  $M$ . Then, a repair for  $M$  given  $\mathbf{KB}$  and  $\mathcal{I}$  is a function  $\mathcal{R} : [1.. \tau] \rightarrow 2^{ABR_M}$  such that there exists a function  $\mathcal{B} : [1.. \tau] \rightarrow \mathbf{Bel}_M$  such that*

- $\mathcal{B}^t$  is an equilibrium of  $M[\mathcal{R}^t]$  given  $\mathcal{KB}^t$  and  $\mathcal{I}^t$ , where  $\mathcal{KB}^t$  is inductively defined as
  - $\mathcal{KB}^1 = \mathbf{KB}$
  - $\mathcal{KB}^{t+1} = \mathbf{upd}_{M[\mathcal{R}^t]}(\mathcal{KB}^t, \mathcal{I}^t, \mathcal{B}^t)$ .

We refer to  $\mathcal{B}$  as a repaired equilibria stream of  $M$  given  $\mathbf{KB}$ ,  $\mathcal{I}$  and  $\mathcal{R}$ .

The introduced notion is very general, as it considers every set of bridge rules whose removal lead to a valid equilibrium to be a *repair*. Therefore this will include these repairs, which would either remove an unnecessary amount of bridge rules or none at all. We will call the latter one an *empty repair* and denote it by  $\mathcal{R}_\emptyset$ .

**Proposition 5.2.27.** *Every equilibria stream of  $M$  given  $\mathbf{KB}$  and  $\mathcal{I}$  is a repaired equilibria stream of  $M$  given  $\mathbf{KB}$ ,  $\mathcal{I}$  and the empty repair  $\mathcal{R}_\emptyset$ .*

*Proof.*  $M[\mathcal{R}_\emptyset] = M$ , therefore the conditions of an equilibria stream (Definition 5.2.16) of  $M$  coincide with the definition of a repaired equilibria stream (Definition 5.2.26) of  $M$ .  $\square$

This result shows that every equilibria stream is a repaired one too, therefore it is viable to use the repaired one instead. Because we have a notion to still allow cycles, we now need to show that total coherence is sufficient to ensure the existence of an repair.

**Proposition 5.2.28.** *Let  $M = \langle \langle C_1, \dots, C_n \rangle, \text{IL}, \text{BR} \rangle$  be a reactive Multi-Context System such that each  $C_i$ ,  $i \in \{1, \dots, n\}$ , is totally coherent,  $\text{KB}$  a configuration of knowledge bases for  $M$ , and  $\mathcal{I}$  an input stream for  $M$  until  $\tau$ . Then, there exists  $\mathcal{R} : [1.. \tau] \rightarrow 2^{br_M}$  and  $\mathcal{B} : [1.. \tau] \rightarrow \text{Bel}_M$  such that  $\mathcal{B}$  is a repaired equilibria stream given  $\text{KB}$ ,  $\mathcal{I}$  and  $\mathcal{R}$ .*

*Proof.* Since each context of  $M$  is totally coherent, Proposition 5.2.25 guarantees the existence of an equilibrium if  $M$  is acyclic. Now just note that if we take  $\mathcal{R} : [1.. \tau] \rightarrow 2^{br_M}$  such that  $\mathcal{R}^t = br_M$  for every  $t$ , then each  $M[\mathcal{R}^t]$  does not have bridge rules and it is therefore acyclic. Then, for every  $t$ ,  $M[\mathcal{R}^t]$  is strongly consistent. Therefore we can easily inductively construct  $\mathcal{B} : [1.. \tau] \rightarrow \text{Bel}_M$  such that  $\mathcal{B}$  is a repaired equilibria stream given  $\text{KB}$ ,  $\mathcal{I}$  and  $\mathcal{R}$ .  $\square$

Total coherence is still a very restricting property and it is only a sufficient one. This can be easily seen, as it disallows every context formalism, which may lead to have no belief set, based on any possible knowledge base. Note that for managed Multi-Context System there is a characterisation based on *omni-coherence* [Weinzierl, 2014] too. The approach uses the idea that total coherence has to hold for every possible combination of bridge rules. Omni-coherence and total coherence are the same, but their inverse is not. Therefore it is possible to say that there is a repair if and only if no omni-incoherent contexts are in place. This approach won't work, because in managed Multi-Context System bridge rules are purely satisfied by the belief state, while in reactive Multi-Context System we have to take the input stream into consideration too.

Repairs allow us to recover the equilibria stream, but we have not seen how we can measure the quality of the repair. In related literature about repairs, e.g. in the context of databases [Arenas et al., 1999], there are always minimal repairs considered to be the most optimal. Intuitively the idea behind that reasoning is that the smallest number of removed elements will change the intuition of the original, but inconsistent, solution the least. As we know that cycles are the cause of inconsistencies this strategy is very viable too, because we are interested in breaking only these cycles which are responsible for the non-existence of an equilibrium.

**Definition 5.2.29.** *Let  $\mathcal{R}_a$  and  $\mathcal{R}_b$  be two repairs for some reactive Multi-Context System  $M$  given a configuration of knowledge bases for  $M$ ,  $\text{KB}$  and  $\mathcal{I}$ , an input stream for  $M$  until  $\tau$ . We say that  $\mathcal{R}_a \leq \mathcal{R}_b$  if  $\mathcal{R}_a^i \subseteq \mathcal{R}_b^i$  for every  $i \leq \tau$ , and that  $\mathcal{R}_a < \mathcal{R}_b$  if  $\mathcal{R}_a \leq \mathcal{R}_b$  and  $\mathcal{R}_a^i \subset \mathcal{R}_b^i$  for some  $i \leq \tau$ .*

We will use this relation to order the possible repairs and check which repairs are minimal. In the following we will discuss different versions of repairs,

which will focus on different ways of optimising repairs over a given input stream.

**Definition 5.2.30** (Types of Repairs). *Let  $\mathcal{R}$  be a repair for some reactive Multi-Context System  $M$  given  $\text{KB}$  and  $\mathcal{I}$ . We say that  $\mathcal{R}$  is a:*

**Minimal Repair** *if there is no repair  $\mathcal{R}_a$  for  $M$  given  $\text{KB}$  and  $\mathcal{I}$  such that  $\mathcal{R}_a < \mathcal{R}$ .*

**Global Repair** *if  $\mathcal{R}^i = \mathcal{R}^j$  for every  $i, j \leq \tau$ .*

**Minimal Global Repair** *if  $\mathcal{R}$  is global and there is no global repair  $\mathcal{R}_a$  for  $M$  given  $\text{KB}$  and  $\mathcal{I}$  such that  $\mathcal{R}_a < \mathcal{R}$ .*

**Incremental Repair** *if  $\mathcal{R}^i \subseteq \mathcal{R}^j$  for every  $i \leq j \leq \tau$ .*

**Minimally Incremental Repair** *if  $\mathcal{R}$  is incremental and there is no incremental repair  $\mathcal{R}_a$  and  $j \leq \tau$  such that  $\mathcal{R}_a^i \subset \mathcal{R}^i$  for every  $i \leq j$ .*

Basically the *minimal Repair* follows the idea of having as few rules removed during a repair step as possible. This corresponds to an ideal solution where no unnecessary bridge rules are kept out during the computation of the equilibria. Nevertheless this might not be always desirable. Maybe one rule is always involved in producing an inconsistency. In this case it would be smart to not use this one rule at all. This consideration leads towards the idea of a *global repair*. Again it would be a refinement over *global repairs* to minimise them too, in order to remove the least amount of always removed rules in the managed Multi-Context System. To compute a *global repair* it is necessary to know the whole input stream in advance. In case we want to decide the repairs on demand, without knowledge of future computations, we can either stick to *minimal repairs* or refine the idea of the *global repair* further. The base intuition behind the *global repairs* is to remove a problematic bridge rule in every time instant. With *incremental repairs* we recycle that idea under the assumption that we cannot analyse the whole input stream. In simple words, we only permit future repairs which repair at least the same as previously. With *minimally incremental repairs* we are now limiting the number of additionally repaired rules to be minimal.

We want to discuss some further ideas, which have been discussed for managed Multi-Context Systems too. There we don't want to repair and remove rules, but make more rules applicable. For managed Multi-Context Systems [Eiter et al., 2014, Brewka et al., 2011b] this is easily doable by just removing literals from the body, or even eliminating the whole body. This is much more complicated for reactive Multi-Context Systems, because we have to keep faithful to the input stream. Following the idea of eliminating the whole body would one allow to assume any input stream information, no matter if it is available or not. It might be a first good idea to limit it only to context

atoms, but then it is questionable if the information of the input stream will not be too biased. One good suggestion would be to make this strengthening of rules only for those which contain no input atoms. This will underline the idea that input streams are definitive while the internal belief states are a subject to change and discussion.

For now we have discussed how to ensure the existence of an equilibria stream in two ways. First we defined sufficient properties to enforce its existence and limited the reactive Multi-Context System very much. Then we released some of the limitations and presented repairs as a way to still deduce some valid equilibrium. We could release the limitation on total coherence with some workaround too. By adding a consistency restoring repair to the management function of a not total coherent context, such that some belief set can be computed for this particular context (e.g. the management function could return  $\{\text{"unsat"}\}$  instead of  $\emptyset$ ). In addition consider that These repairs, while easily defined, are very complex to compute on demand, because each repair candidate needs to be tested if it is a valid repair. When using more sophisticated repairs, like a global, incremental, or minimised approach it will need even more computations to find the right candidate. So even under the existence of ways to work around cyclic and totally incoherent contexts, one might still not be intrigued to use that much computational effort to get the system consistent all the time.

One solution to that problem is to relax the notion of an equilibria stream<sup>35</sup>. In detail we want to remove the necessity of an equilibrium in every time instant and allow the stream of equilibria to be undefined at certain points.

**Definition 5.2.31** (Partial Equilibria Stream). *Let  $M = \langle C, IL, BR \rangle$  be an reactive Multi-Context System,  $KB = \langle kb_1, \dots, kb_n \rangle$  a configuration of knowledge bases for  $M$ , and  $\mathcal{I}$  an input stream for  $M$  until  $\tau$  where  $\tau \in \mathbb{N} \cup \{\infty\}$ . Then, a partial equilibria stream of  $M$  given  $KB$  and  $\mathcal{I}$  is a partial function  $\mathcal{B} : [1..\tau] \rightarrow \text{Bel}_M$  such that*

- $\mathcal{B}^t$  is an equilibrium of  $M$  given  $\mathcal{KB}^t$  and  $\mathcal{I}^t$ ,
- or  $\mathcal{B}^t$  is undefined.

$\mathcal{KB}^t$  is inductively defined as

- $\mathcal{KB}^1 = KB$
- $\mathcal{KB}^{t+1} = \begin{cases} \text{upd}_M(\mathcal{KB}^t, \mathcal{I}^t, \mathcal{B}^t), & \text{if } \mathcal{B}^t \text{ is not undefined.} \\ \mathcal{KB}^t, & \text{otherwise.} \end{cases}$

---

<sup>35</sup>Note that the idea of relaxing the notion of an equilibria is not new. In their work [Dao-Tran et al., 2015] the authors relaxed the notion of an equilibrium by only computing the equilibrium over a set of contexts instead of all.

Of course this definition is chosen in such a way that every stream of equilibria is a partial equilibria stream as well. Therefore it is a proper generalisation of the notion of equilibria streams.

**Proposition 5.2.32.** *Every equilibria stream of  $M$  given  $\mathcal{KB}$  and  $\mathcal{I}$  is a partial equilibria stream of  $M$  given  $\mathcal{KB}$  and  $\mathcal{I}$ .*

*Proof.* This result follows easily from the observation that for an equilibria stream  $\mathcal{B}$  of  $M$  given  $\mathcal{KB}$  and  $\mathcal{I}$ , and every  $t$ ,  $\mathcal{B}^t$  is never undefined. Therefore, in this case the conditions in the definition of partial equilibria stream coincide with those for equilibria stream.  $\square$

In contrast to equilibria streams, we can now ensure that under every circumstance a partial equilibria stream exists. This enables one to relax all of the previously discussed properties and mechanics for the price of having the possibility to undefined and therefore non existent equilibria.

**Proposition 5.2.33.** *Let  $M$  be an reactive Multi-Context System,  $\mathcal{KB}$  a configuration of knowledge bases for  $M$ , and  $\mathcal{I}$  an input stream for  $M$  until  $\tau$ . Then, there exists  $\mathcal{B} : [1..\tau] \rightarrow \text{Bel}_M$  such that  $\mathcal{B}$  is a partial equilibria stream given  $\mathcal{KB}$  and  $\mathcal{I}$ .*

*Proof.* We just need to note that if we take  $\mathcal{B} : [1..\tau] \rightarrow \text{Bel}_M$  such that, for every  $t$ ,  $\mathcal{B}^t$  is undefined, then  $\mathcal{B}$  is trivially a partial equilibria stream given  $\mathcal{KB}$  and  $\mathcal{I}$ .  $\square$

Note that a partial equilibria stream is not only useful for (acyclic) reactive Multi-Context Systems which have totally incoherent contexts. This concept becomes handy in other settings too. Sometimes it might be unwanted to compute the equilibria at every time instant. That could be because no new information arrived via the input stream and no further computation is wanted. Another reason is that the computation of the equilibrium could not be finished in a set amount of time. Of course we can strengthen the notion of partial equilibria streams to force the reactive Multi-Context System to finish the computation and only relax equilibria streams if it is impossible to compute a equilibrium. To accomplish that, we need to refine the definition of  $\mathcal{B}^t$  in Definition 5.2.31, such that  $\mathcal{B}^t$  is undefined if and only if there is no equilibrium of  $M$  given  $\mathcal{KB}$  and  $\mathcal{I}$ .

During this section we have only considered declarative bridge rules, because we were only interested in computing a valid equilibrium. From the formal point of view this discussions are completely sound, but due to the decoupled nature of operative and declarative bridge rules it might lead to unintended side effects with respect to repairs. Intuitively reactive Multi-Context Systems are designed in such a way that the declarative bridge rules allow, with the notion of an equilibrium, a setting where each context needs to agree towards the beliefs and their caused temporal changes to their knowledge bases. On basis

of this agreement the operational bridge rules will perform changes on behalf of the agreed beliefs. In other words, the operational changes are approved by the equilibrium, computed by the declarative bridge rules and the contexts. Repairs are designed to remove declarative bridge rules which cause inconsistencies and make it impossible for the contexts to agree on any result. They do not take into consideration that the operational bridge rules are not aware of the removal of some, or even all declarative rules. Because the two rule types are totally distinct from each other, it is not possible to relate every operational rule to a declarative one. That can be seen in the example scenario of Section 5.2.3. There are contexts which have operational rules which are totally unrelated to the declarative ones (e.g. the stove contexts rules for reacting on the power switch).

Nevertheless we want to present an idea on how to solve this problem for these rules which rely on the reasoning from the equilibrium computation. We will make a connection between the operational and the declarative bridge rules, such that the operational rule can only be satisfied if the declarative has been triggered beforehand. To model this directly in the framework of a reactive Multi-Context System, we will adapt all the operators such that they are aware of their calling bridge rule. The management function will be changed accordingly to add a list of all the triggered bridge rules into its knowledge base. In addition each declarative bridge rule gets its own unique designation with respect to the context. Then each bridge rules body gets an additional positive context literal which, namely that the rule can only be applied if the context believes that the rule has been triggered. Now it is easy to enhance each dependant operational rule such that it only is utilised if the context believed in the equilibrium that the needed declarative rules have been used.

### 5.2.5 Limiting Non-Deterministic Effects

The previous section has discussed on how to ensure the existence of at least one equilibrium during each computation step. This section will focus on a slightly different problem, namely the reduction of possible equilibria, which has been considered before in an early stage of reactive Multi-Context System [Ellmauthaler, 2013]. It is obvious that more than one belief states are valid equilibria and due to the definitions of the reactive Multi-Context System, it is one source of non-determinism that it is not defined which equilibrium is chosen for the equilibria stream. Though, it is not given that each equilibrium has the same amount of quality, with respect to the other valid candidates.

To achieve this goal, some kind of measurement will be needed to find the best equilibrium. Due to the need of a comparison of different computations, it is not possible to just use some modelling techniques, like we have done it for other things before. Therefore the introduction of preferences is a straightforward

solution to this issue. Still, it is not entire clear how these preferences should be added. Some viable options are

- (i) adding weights to the declarative bridge rules and optimise the weights,
- (ii) allowing the contexts to rate their result and optimise the individual or global score, or
- (iii) adding preference relations over sets of belief sets,

which offer different kind of advantages. By using option (i), it is pretty easy and straight forward to just sum up the weights and only allow these equilibria which have either the minimal or the maximal value in total. Basically that is an easy approach, but on the other hand it might be a tricky task to foresee the impact of the usage of one operator, based on the triggered bridge rule. In addition it might become challenging to properly prefer combinations of used bridge rules over others. Option (ii), which allows each context to give its (possibly weighted) score for the found equilibrium makes it very intuitive. Every context has to agree to the belief state, so it is not very awkward that the context can state how optimal that solution is from its point of view. This approach can simulate the former one too, by letting the operators and the management function track which rule has triggered the changes and add this information to the context. In contrast to that, option (iii) will prefer some answers over others on the global level. The biggest disadvantage of this concept is that it would need many different preferences to impose an ordering of the different equilibria. Additionally it this information needs to be changed every time some contexts or rules are changed.

Because of the above discussion we conclude that option (ii) seems to be an easy to adapt and easy to use way to introduce preferences and optimisation to reactive Multi-Context Systems. The evaluation of the equilibrium is done by every context in an independent manner and it is possible to easily add weights to each context. This approach can take information about bridge rule usage, knowledge on the computation of the equilibria from the point of view of each context, and meta-knowledge about the different contexts into account.

### 5.2.6 Expressiveness

Here we will show that our approach is at least as expressive as an universal Turing machine. This will be done by illustrating a way to simulate the working processes of a Turing machine with a reactive Multi-Context System. To show that the concepts of the reactive Multi-Context System are doing the simulation we will only use a simple logic and the management-function will only perform basic operations on sets (i.e. set difference  $\setminus$  and union  $\cup$ ). That means the computation of the states of the Turing machine are solely handled by the interaction of bridge rules and their corresponding equilibria stream. We use

the definition of a Turing machine, already presented in Section 3.2, which is recapitulated below.

**Definition 3.2.1.** *A TM is a tuple  $\langle Q, \Gamma, \sqcup, \Sigma, \delta, q_0, F \rangle$ , where*

- $Q \subseteq \mathbb{Q}$  is a finite, non-empty set of states,
- $\Gamma \subseteq \mathbb{S}$  is a finite, non-empty set of tape symbols, the alphabet,
- $\sqcup \in \Gamma$  is the blank symbol,
- $\Sigma$  is a sequence of alphabet symbols, the input,
- $q_0$  is the initial state,
- $F \subseteq Q$  is the set of final states, and
- $\delta : Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$  is the (partial) transition function.

$M_{\text{TM}}$  is a reactive Multi-Context System with four contexts. Context  $C_t$  simulates a tape of a TM,  $C_q$  contains information about TM states,  $C_f$  encodes a transition function, and  $C_c$  is a control context for operating the TM simulation and presenting results. All contexts use a storage logic as in Example 5.2.1, where the set of entries is respectively given by

- $E_t = \bigcup_{p \in \mathbb{Z}, s \in \mathbb{S}} \{t(p, s), \text{curP}(p)\}$ ,
- $E_q = \bigcup_{q \in \mathbb{Q}} \{\text{final}(q), \text{curQ}(q)\}$ ,
- $E_f = \{f(q, s, q', s', m) \mid q, q' \in \mathbb{Q}, s, s' \in \mathbb{S}, m \in \{L, R\}\}$ , and
- $E_c = \{\text{computing}, \text{answer}(yes), \text{answer}(no)\}$ .

The input of  $M_{\text{TM}}$  is provided by four input streams over the languages

- $IL_t = E_t$ ,
- $IL_q = E_q$ ,
- $IL_f = E_f$ , and
- $IL_c = \{\text{start}, \text{reset}\}$ ,

where  $IL_t$  allows for setting an initial configuration for the tape,  $IL_q$  the allowed and final states of the TM to simulate,  $IL_f$  the transition function, and input over  $IL_c$  is used to start and reset the simulation.



The bridge rule schemata of the tape context  $C_t$  are given by:

$$\begin{aligned}
 \mathbf{next}(\mathbf{add}(t(P, S'))) &\leftarrow q:\mathbf{f}(Q, S, Q', S', D), t:\mathbf{curP}(P), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), S \neq S', c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{rm}(t(P, S))) &\leftarrow q:\mathbf{f}(Q, S, Q', S', D), t:\mathbf{curP}(P), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), S \neq S', c:\mathbf{computing}. \\
 \mathbf{add}(\mathbf{nextP}(P - 1)) &\leftarrow q:\mathbf{f}(Q, S, Q', S', \leftarrow), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), t:\mathbf{curP}(P), c:\mathbf{computing}. \\
 \mathbf{add}(\mathbf{nextP}(P + 1)) &\leftarrow q:\mathbf{f}(Q, S, Q', S', \rightarrow), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), t:\mathbf{curP}(P), c:\mathbf{computing}. \\
 \mathbf{add}(\mathbf{nextPdefined}) &\leftarrow t:\mathbf{nextP}(P), c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{add}(\mathbf{curP}(P))) &\leftarrow t:\mathbf{nextP}(P), c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{rm}(\mathbf{curP}(P))) &\leftarrow t:\mathbf{curP}(P), t:\mathbf{nextPdefined}, c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{add}(X)) &\leftarrow t::X, \mathbf{not} c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{clear}) &\leftarrow c::\mathbf{reset}.
 \end{aligned}$$

The bridge rule schemata for the state context  $C_q$  are the following:

$$\begin{aligned}
 \mathbf{next}(\mathbf{add}(\mathbf{curQ}(Q'))) &\leftarrow q:\mathbf{f}(Q, S, Q', S', D), t:\mathbf{curP}(P), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), Q \neq Q', c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{rm}(\mathbf{curQ}(Q))) &\leftarrow q:\mathbf{f}(Q, S, Q', S', D), t:\mathbf{curP}(P), q:\mathbf{curQ}(Q), \\
 &\quad t:t(P, S), Q \neq Q', c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{add}(X)) &\leftarrow q::X, \mathbf{not} c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{clear}) &\leftarrow c::\mathbf{reset}.
 \end{aligned}$$

The state  $C_f$  for the transition function has the bridge rules schemata given next:

$$\begin{aligned}
 \mathbf{next}(\mathbf{add}(X)) &\leftarrow f::X, \mathbf{not} c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{clear}) &\leftarrow c::\mathbf{reset}.
 \end{aligned}$$

Finally, the schemata for  $C_c$  are:

$$\begin{aligned}
 \mathbf{add}(\mathbf{answer}('Y')) &\leftarrow q:\mathbf{curQ}(Q), q:\mathbf{finalQ}(Q), c:\mathbf{computing}. \\
 \mathbf{add}(\mathbf{answer}('N')) &\leftarrow \mathbf{not} t:\mathbf{nextPdefined}(Q), q:\mathbf{curQ}(Q), \\
 &\quad \mathbf{not} q:\mathbf{finalQ}(Q), c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{add}(\mathbf{answer}(X))) &\leftarrow c:\mathbf{answer}(X), c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{rm}(\mathbf{computing})) &\leftarrow c:\mathbf{answer}(X), c:\mathbf{computing}. \\
 \mathbf{next}(\mathbf{clear}) &\leftarrow c::\mathbf{reset}. \\
 \mathbf{next}(\mathbf{add}(\mathbf{computing})) &\leftarrow c::\mathbf{start}.
 \end{aligned}$$

All contexts use the following management function:

$$\mathbf{mng}(OP, kb) = \begin{cases} \emptyset & \text{if } \mathbf{clear} \in OP \\ kb \setminus \{X \mid \mathbf{rm}(X) \in OP\} \cup \{X \mid \mathbf{add}(X) \in OP\} & \text{else} \end{cases}$$

Let  $T = \langle Q, \Gamma, \cup, \Sigma, \delta, q_0, F \rangle$  be a TM and  $w \in \Sigma^*$  an input word for  $T$ . We want to use  $M_{\text{TM}}$  with input stream  $\mathcal{I}$  to simulate  $T$ . Assume we start at time  $t$ . We first make sure that all knowledge bases are empty by setting  $\mathcal{I}_c^t = \{\text{reset}\}$ . This activates the bridge rules in all contexts of  $M_{\text{TM}}$  that derive **next(clear)**. As a consequence, at time  $t + 1$  the contents of all knowledge bases are deleted. Next, we feed  $T$  and  $w$  to  $M_{\text{TM}}$  by sending

- $\text{final}(q)$  for all  $q \in F$  and  $\text{curQ}(q_0)$  on the input stream  $q$ ,
- $\text{f}(q, s, q', s')$  iff  $\delta(q, s) = \langle q', s' \rangle$  on stream  $f$ , and
- $\text{curP}(\theta)$  and  $\text{t}(p, s)$  iff  $s = s_p$  for  $w = s_0, s_1, s_2, \dots$  on the tape stream  $t$ .

Note that it does not matter whether we do this all at once at time  $t + 1$  or scattered over multiple time points greater than  $t$ . Assume that we finished to incorporate all this information to the knowledge bases at time  $t'$ . Then, we set  $\mathcal{I}_c^j = \{\text{start}\}$  to initiate the simulation of  $T$ . At time  $t' + 1$  the entry computing is contained in the knowledge base of context  $C_c$ , activating the bridge rules in all contexts that are responsible for the simulation. From now on, depending on the current state  $\text{curQ}(q)$  and the transition function, the bridge rules of tape context  $C_t$  always change the content of the tape on the current tape position indicated by  $\text{curP}(p)$ . A new position  $p'$  of the tape head indicated by the transition function is reflected by deriving  $\text{nextP}(p')$ . If such a belief is in the equilibrium so is  $\text{nextP}$  defined and  $\text{curP}(p')$  is added at the next time point. For context  $C_q$  the current state is updated according to the transition function. Note that the auxiliary belief  $\text{nextP}$  defined is also used in the bridge rules of context  $C_c$  for indicating that if the current state is not final and the transition function is undefined for the current state and input symbol, then the answer of the TM is no, indicated by  $\text{answer}('N')$ . Conversely, if we arrive at a final state then  $\text{answer}('Y')$  is derived. If  $T$  does not halt on input  $w$ , then also the simulation in  $M_{\text{TM}}$  will continue forever, unless we stop the computation by sending  $\text{reset}$  on input stream  $\mathcal{I}_c$  once more.

### 5.2.7 Complexity

In this part of the section we will conclude the discussion on reactive Multi-Context Systems with a complexity analysis of the introduced theory. Strictly speaking we are interested in the complexity of answering queries over equilibria streams of a reactive Multi-Context System. We will stick to the case where the input is finite, because in the infinite case the decision problem turns out to be undecidable (see Proposition 5.2.35). Therefore we will consider a reactive Multi-Context Systems with a finite input stream, such that  $\tau \in \mathbb{N}$ . In addition we will not consider rule schemata, because they are capable of generating an infinite amount of rules in general. The third restriction we impose is, that every knowledge base for the contexts of the reactive Multi-Context System as well as every input is finite. We will call a reactive Multi-Context System *finite*

if these two confinements on knowledge bases and inputs hold. We are also assuming that the management function is working in  $\mathbf{P}$ . Note that we need to introduce these limitations because the framework of reactive Multi-Context Systems offers many different degrees of freedom and we are mostly interested in the computational complexity of the core mechanics and ideas of the framework. To define a specific set of decision problems, we will tackle two different ones, which are commonly used for managed Multi-Context Systems.

**Definition 5.2.34** (Decision Problems). *The problem  $Q^\exists$ , respectively  $Q^\forall$ , is deciding whether for a given finite reactive Multi-Context System  $M$ , a belief  $b$  for the  $k$ -th context of  $M$ , a configuration of knowledge bases  $\mathbf{KB}$  for  $M$ , and an input stream  $\mathcal{I}$  until  $\tau$ , it holds that  $b \in B_k$  for some  $\mathcal{B}^t = \langle B_1, \dots, B_n \rangle$ , ( $1 \leq t \leq \tau$ ), for some, respectively all, equilibria stream(s)  $\mathcal{B}$  given  $\mathbf{KB}$  and  $\mathcal{I}$ .*

Intuitively these two decision problems cover the question of whether some belief might hold at some time in at least equilibria stream or if it can be deduced in every equilibria stream. It is important to note that for these settings it is not important at which time instant that belief has been holding.

**Proposition 5.2.35.** *Given a finite reactive Multi-Context System  $M$ , the problems  $Q^\exists$  and  $Q^\forall$  are undecidable for infinite input streams (when  $\tau = \infty$ ).*

*Proof.* In Section 5.2.6 it is shown that a reactive Multi-Context System with simple context logics can simulate a Turing machine. Deciding  $Q^\forall$  or  $Q^\exists$  for an infinite run would solve the halting problem, therefore these problems need to be undecidable too.  $\square$

To speak about the complexity of the whole reactive Multi-Context System, we will need to take the complexity of the contexts too. We will use the notion of *context complexity* (CC 5.2.37), first used in analysing Multi-Context System [Eiter et al., 2014]. For a definition of the context complexity, we will only focus on the relevant parts of the bridge rules, which will decide  $Q^\exists$  or  $Q^\forall$ .

**Definition 5.2.36** (Relevant Beliefs). *Given  $M$ ,  $b$ ,  $k$ , and  $\mathcal{I}$  as in Definition 5.2.34, the set of relevant beliefs for a context  $C_i$  of  $M$  is given by*

$$RB_i(M, k:b) = \{b' \mid r \in BR_h, i:b' \in \text{body}(r) \vee \mathbf{not} \ i:b' \in \text{body}(r), \\ h \in \{1, \dots, n\}\} \cup \\ \{b \mid k = i\}.$$

*Then, a projected belief state for  $M$  and  $k:b$  is a tuple*

$$\mathbf{B}_{|M}^{k:b} = \langle B_1 \cap RB_1(M, k:b), \dots, B_n \cap RB_n(M, k:b) \rangle$$

*where  $\mathbf{B} = \langle B_1, \dots, B_n \rangle$  is a belief state for  $M$ . If  $\mathbf{B}$  is an equilibrium, then we call this tuple projected equilibrium.*

$\mathcal{CC}(M, k:b)$	$Q^\exists$	$Q^\forall$
<b>P</b>	<b>NP</b>	<b>coNP</b>
$\Delta_i^P (i \geq 2)$	$\Sigma_i^P$	$\Pi_i^P$
$\Sigma_i^P (i \geq 1)$	$\Sigma_i^P$	$\Pi_i^P$
<b>PSPACE</b>	<b>PSPACE</b>	<b>PSPACE</b>
<b>EXPTIME</b>	<b>EXPTIME</b>	<b>EXPTIME</b>

Table 5.1: Complexity results of checking  $Q^\exists$  and  $Q^\forall$ 

Now we can formulate the context complexity, which is defined for each context individually and for a given reactive Multi-Context System. Note that for the complexity of the reactive Multi-Context System, we will only need to take a look at the highest complexity context, as their computations are not dependent on the computation of the other contexts.

**Definition 5.2.37** (Context Complexity). *The context complexity of  $C_i$  in  $M$  with respect to  $k:b$  for a fixed input  $l$  is the complexity of deciding the context problem of  $C_i$ , that is, whether for a given projected belief state  $\mathbf{B} = \langle B_1, \dots, B_n \rangle$  for  $M$  and  $k:b$ , there is some belief set  $B'_i$  for  $C_i$  with  $B_i = B'_i \cap RB_i(M, k:b)$  and  $B'_i \in \mathbf{acc}_i(\mathbf{mng}_i(\mathbf{app}_i(l, \mathbf{B}), kb_i))$ . The context complexity  $\mathcal{CC}(M, k:b)$  of an entire reactive Multi-Context System is a (smallest) upper bound for the context complexity classes of its contexts.*

Table 5.1 is now specifying the complexity for both decision problems under a given context complexity for a given finite reactive Multi-Context System. Note that this shows pretty easily that the overall complexity always depends on the most complex context formalism and might move up to one level in the polynomial hierarchy. This is to be expected, considering how the equilibrium is computed. While this behaviour is unsurprisingly, it has some conclusions to consider. If one context has a high computational complexity compared to the others (e.g. description logic has decision problems in **EXPTIME**, while answer set programming is usually at most at  $\Sigma_2^P$ ), the whole system might need to wait for the computation of the belief set of one of the computational harder to compute problems. Of course it might not be as severe as it sounds like, as these complexities are worst case scenarios and in general one can try to utilise easier to compute sub-classes of the problems or use intelligent pre-processing steps and algorithms, but it is still something to consider.

**Theorem 5.2.38.** *Table 5.1 summarizes the complexities of membership of problems  $Q^\exists$  and  $Q^\forall$  for finite input streams (until some  $\tau \in \mathbb{N}$ ) depending on the context complexity. Hardness also holds if it holds for the context complexity.*

*Proof.* The membership results for the  $Q^{\exists}$  cases (with the exception of the case where  $\mathcal{CC}(M, k:b) = \mathbf{EXPTIME}$ ) can be argued for as follows: a non-deterministic Turing machine can be used to guess a projected belief state  $\mathcal{B}^t = \langle B_1, \dots, B_n \rangle$  for all  $\tau$  inputs in  $\mathcal{I}$  in polynomial time. Then, iteratively for each of the consecutive inputs  $\mathcal{I}^t$ , first the context problems can be solved either polynomially or using an oracle for the context complexity (the guess of  $\mathcal{B}^t$  and the oracle guess can be combined which explains why we stay on the same complexity level for higher context complexity). If the answer is 'yes',  $\mathcal{B}^t$  is the projected equilibrium. We can check whether  $b \in B_i$ , compute the updated knowledge bases and continue the iteration until reaching the last input. For  $\mathbf{PSPACE}$  the same line of argumentation holds as  $\mathbf{PSPACE} = \mathbf{NPSPACE}$ . In the case of  $\mathcal{CC}(M, k:b) = \mathbf{EXPTIME}$ , we iterate through the exponentially many projected belief states for which we solve the context problem in exponential time and proceed as before. The argument is similar for the co-problem of  $Q^{\forall}$ . Hardness holds because being able to solve  $Q^{\exists}$ , respectively the co-problem of  $Q^{\forall}$ , one can decide equilibrium existence for managed MCSs which is hard for the same complexity classes [Brewka et al., 2011b] given hardness for the context complexity of the managed Multi-Context System.  $\square$



---

# Asynchronous Multi-Context Systems

---

Now we want to introduce and discuss another framework for Multi-Context Systems. The concept of asynchronous Multi-Context Systems has already been considered in the paper about ways to generalise Multi-Context Systems towards stream reasoning [Ellmauthaler, 2013]. There only the idea of “*reactive Bridge Rules*” has been introduced and discussed. Intuitively the idea was to move away from the strong semantics of managed Multi-Context Systems and allowing these kind of rules to exchange their beliefs based on their own beliefs without the need of approval of other contexts. Further research on this basic concept has led towards the introduction of asynchronous Multi-Context Systems [Ellmauthaler and Pührer, 2014, Ellmauthaler and Pührer, 2015] and later it got refined by a way to control the flow of information even more in terms of information packaging [Ellmauthaler and Pührer, 2016].

The motivation for developing asynchronous Multi-Context Systems was based on three different observations.

First, reactive Multi-Context Systems are a very powerful formalism to model integration of information between different context formalisms and means to react dynamically to external data over time. Nevertheless, all the contexts need to do their reasoning together and therefore every embedded formalism is kind of synchronised to each other. This fact is reflected by the overall computational complexity of reactive Multi-Context Systems too (see Section 5.2.7). In addition each belief state which needs to be checked for being an equilibrium needs to be communicated to every context beforehand, which might lead to a vast amount of communication overhead. Here the idea of a paradigm shift emerged, which allows each context to compute its belief sets on its own pace and inform the others about the results. Due to this basic different and more loosely coupled, asynchronous approach the framework has been named asynchronous Multi-Context System.

The second motivational reason is a severe lack of some easy to use concept

to define exchange of information between intelligent systems in a dynamic environment. We tried to design these asynchronous Multi-Context System in such a way that it consists out of different modules which can be defined as independently as possible. Therefore we created a way to define the whole information flow, the computation of beliefs, and reasoning on behalf of them in a uniform pattern. This allows one to settle a basic ground for definition, testing, and evaluation of systems which are tailored towards knowledge integration as well as reasoning in dynamic environments.

Third, it is not very common in the current world of the world wide web, that information is synchronously spread during computations of different services. It is more common that requests and responses are sent in between the different services to transfer information (e.g. RESTful web services [Richardson and Ruby, 2008]). So it is a reasonable step to change the flow of information such that the information is sent towards other contexts, which are interested in the given data. In other words, we don't want the information being spread out to every context, but delivered to these contexts which need it.

Note that we want to introduce asynchronous Multi-Context Systems as a language formalism to standardise communication and knowledge exchange between intelligent reasoning contexts. This approach has enough expressiveness to model the integration of knowledge between different formalisms and can still handle the dynamic responses needed for reasoning over time with external information. In addition it allows for *loosely coupled* reasoning between contexts to reduce communication overhead and neglecting the need of synchronous reasoning. Still, the framework will be able to have groups of contexts which behave like a reactive Multi-Context System, which will be shown later in this chapter.

The remainder of this chapter is structured as follows: First we will introduce asynchronous Multi-Context Systems in Section 6.1. Then we will give a brief modelling example, based on the second scenario given in Chapter 2<sup>36</sup>. To relate the whole concept to reactive Multi-Context Systems, a simulation approach will be presented in Section 6.3 and finally we will discuss a method to manage the incoming information of a context via a technique called *declarative data set packing*.

## 6.1 Syntax and Semantics of Asynchronous Multi-Context Systems

For the formalism used inside the contexts, we will utilise the more expressive and adaptive version of a logic suite (see Section 3.4 for further details). For easier readability we recall the previously given Definition 3.4.1 again.

---

<sup>36</sup>Note that this example is only marginally updated from the one given in [Ellmauthaler and Pührer, 2015] and some passages are taken directly from my own work done there



**Definition 3.4.1.** A logic suite  $LS = (KB_{LS}, BS_{LS}, ACC_{LS})$  consists of the set  $BS_{LS}$  of possible belief sets, the set  $KB_{LS}$  of well-formed knowledge-bases, and a nonempty set  $ACC_{LS}$  of possible semantics of  $LS$ , i.e.  $\mathbf{acc}_{LS} \in ACC_{LS}$  implies  $\mathbf{acc}_{LS} : KB_{LS} \rightarrow 2^{BS_{LS}}$ .

An asynchronous Multi-Context System will use the context of sensors as external incoming information streams and output streams to deliver concluded information to the outside world. We will assume a set  $N$  of names that will act as labels for sensors, contexts, and output streams in a given asynchronous Multi-Context System. Note that the naming will be under the unique name assumption, which means that two different individuals will have different names.

**Definition 6.1.1** (Asynchronous Context). A context is a pair  $C = \langle \mathbf{n}, LS \rangle$  where  $\mathbf{n} \in N$  is the name of the context and  $LS$  is a logic suite.

In analogy to previous notations we will use  $\mathbf{n}_C$  and  $LS_C$  respectively to denote the two members of a context  $C = \langle \mathbf{n}, LS \rangle$ . With the contexts defined, we can now get an understanding of an asynchronous Multi-Context System.

**Definition 6.1.2** (Asynchronous Multi-Context System). An asynchronous Multi-Context System (of length  $n$  with  $m$  output streams) is a pair  $M = \langle \mathbf{C}, \mathbf{O} \rangle$ , where  $\mathbf{C} = \langle C_1, \dots, C_n \rangle$  is an  $n$ -tuple of contexts and  $\mathbf{O} = \langle \mathbf{o}_1, \dots, \mathbf{o}_m \rangle$  with  $\mathbf{o}_j \in N$  for each  $1 \leq j \leq m$  is a tuple containing the names of the output streams of  $M$ .

To denote the names used in the set  $\{\mathbf{n}_{C_1}, \dots, \mathbf{n}_{C_n}, \mathbf{o}_1, \dots, \mathbf{o}_m\}$ , we will write  $N(M)$ , which is a collection of all names used in an asynchronous Multi-Context System for contexts and output streams. One basic idea behind the asynchronous Multi-Context Systems is to model communication with other contexts and the outside world by means of streams of data. To do so, we will assume that every context has an input stream on which information can be written from external sources which we call sensors and internal sources (i.e. other contexts). These input streams are populated by some pieces of abstract information  $i \in IL$  which is part of some communication language  $IL$ . In our framework, the handled information is provided towards the different contexts and output streams via information buffers which model the incoming information.

**Definition 6.1.3** (Data Package). A data package is a pair  $D = \langle \mathbf{s}, I \rangle$ , where  $\mathbf{s} \in N$  is either a context name or a sensor name, stating the source of  $D$ , and  $I \subseteq IL$  is a set of pieces of information. An information buffer  $\mathbf{IB}$  is a sequence of data packages.

Intuitively we assume that this information buffer is filled on the fly, whenever some abstract information in form of a data package is arriving. This will of course happen in an asynchronous way, as we do not have any synchronisation

or ordering process for them. Due to this asynchronous handling of information, our system needs a way to decide whether important information for further processing is still missing or enough data has arrived already. This decision is done by the computation controller.

**Definition 6.1.4** (Computation Controller). *Let  $C = \langle n, LS \rangle$  be a context, and  $SIB$  the set of all possible finite information buffers.. A computation controller for  $C$  is a function  $cc : KB_C \times SIB \rightarrow \{0, 1\}$  mapping either true or false to every possible knowledge base  $kb \in KB$  and information buffer  $IB \in SIB$ .*

In this presented version the function computes only a binary decision, namely whether the information of the input buffer has enough information for computation or a computation should be postponed till more information is available. Of course it would be easy to add additional decisions for this function, by allowing additional values which are representing that a computation should be cancelled and started with new information or that the system should pause and postpone some computational work. Depending on the used formalism, another extension might be to allow some late delivery of additional information<sup>37</sup>.

Next we will have a look on how to transfer beliefs between contexts. For reactive Multi-Context Systems we have used bridge rules, but these are not applicable because we do not want to construct some notions of an equilibrium. Therefore we will use *output rules*, which will only allow atoms with respect to their own context. Note that a rule will be satisfied based on the independently computed beliefs of this one context. The basic idea is now, that the head of such a rule will define to which other buffer what pieces of constructed information are sent to.

**Definition 6.1.5** (Output Rule). *Let  $C = \langle n, LS \rangle$  be a context. An output rule  $r$  for  $C$  is an expression of the form*

$$\langle t, i \rangle \leftarrow b_1, \dots, b_j, \mathbf{not} b_{j+1}, \dots, \mathbf{not} b_m, \quad (6.1)$$

*such that  $t \in N$  is the name of a context or an output stream,  $i \in IL$  is a piece of information, and every  $b_k$  ( $1 \leq k \leq m$ ) is a belief for  $C$ , i.e.  $b_k \in B$  for some  $B \in BS_{LS}$ .*

To refer to these buffers which get information delivered from one context, we say that  $t$  is a *stakeholder* of  $n$ . So intuitively, a stakeholder is an address reference to the receiver of some information  $i$ . In addition we will use the notions of *head*( $r$ ) and *body*( $t$ ) to refer to the head and body part in the same way as it has been done for reactive Multi-Context System.

---

<sup>37</sup>For some formalisms concepts of lazy or on-demand reasoning are currently developed, like incremental clingo [Gebser et al., 2011e] or online clingo [Gebser et al., 2012a] for Answer Set Programming encodings.

Now it might get useful to have means to know which information gets delivered from one context to one other stakeholder. This is defined by the *relevant output* of a context with respect to the set of output rules and the computed belief sets of the context.

**Definition 6.1.6** (Output). *Let  $C = \langle n, LS \rangle$  be a context,  $OR$  a set of output rules for  $C$ ,  $B \in BS_{LS}$  a belief set, and  $n' \in N$  a name. Then, the data package*

$$d_C(B, OR, n') = \langle n, \{i \mid r \in OR, head(r) = \langle n', i \rangle, B \models body(r)\} \rangle$$

*is the output of  $C$  with respect to  $OR$  under  $B$  relevant for  $n'$ .*

In the same manner as it has been done for the revised reactive Multi-Context Systems, we still have no knowledge base defined for the different contexts of the asynchronous Multi-Context System. For the same reasons of dynamics over time, we have chosen to define them outside the base formalism. We will wrap this concept, together with computation control, translation from input buffer to knowledge bases, and passing information through to other stakeholders into the notion of a *context configuration*.

**Definition 6.1.7** (Configuration of a Context). *Let  $C = \langle n, LS \rangle$  be a context. A configuration of  $C$  is a tuple  $CF = \langle kb, \mathbf{acc}, \mathbf{IB}, \mathbf{CM} \rangle$ , where  $kb \in BS_{LS}$ ,  $\mathbf{acc} \in ACC_{LS}$ ,  $\mathbf{IB}$  is a finite information buffer, and  $\mathbf{CM}$  is a context management for  $C$  which is a triple  $\mathbf{CM} = \langle \mathbf{cc}, \mathbf{cu}, OR \rangle$ , where*

- $\mathbf{cc}$  is a computation controller for  $C$ ,
- $OR$  is a set of output rules for  $C$ , and
- $\mathbf{cu}$  is a context update function for  $C$  which is a function that maps an information buffer  $\mathbf{IB} = \langle D_1, \dots, D_m \rangle$  and an admissible knowledge base of  $LS$  to a configuration  $CF' = \langle kb', \mathbf{acc}', \mathbf{IB}', \mathbf{CM}' \rangle$  of  $C$  with  $\mathbf{IB}' = \langle D_k, \dots, D_m \rangle$  for some  $k \geq 1$ .

Again, we will use  $\mathbf{cc}_{CM}$ ,  $OR_{CM}$ , and  $\mathbf{cu}_{CM}$  to refer to the components of the context management. Note that this notion of a context management can be seen as a more powerful counterpart to the management function used for reactive Multi-Context System. One configuration holds the current knowledge base, the currently chosen semantics for the context formalism, and the currently accumulated input stream. In addition it has the context management, whose purpose it is to modify the context based on the given information of the input buffer. First it can be decided via the computation controller if something needs to be done. The context update function allows the context management to produce a new configuration based on the current knowledge base and the input buffer. That allows the whole context to change over time to react to the dynamics of the whole asynchronous Multi-Context System. In the easiest version the context update function will only remove the

information from the input buffer which has been taken into account already and is translating the input buffer data into updates to the knowledge base or the used context semantics. The latter one can become particularly handy, because it might happen that two problems might be solved by the same semantics, but a specialised algorithm or variation of the semantics might perform better on one of the problems. Due to the high adaptive nature of this formalism, we think that fast on-demand change of semantics makes much more sense than for reactive Multi-Context Systems. Still, it is possible to do even more, like changing the output rules to register or remove stakeholders, change the quantity and quality of information given to them, and even change the computation controllers behaviour.

**Definition 6.1.8** (Configuration of an Asynchronous Multi-Context System). *Let  $M = \langle\langle C_1, \dots, C_n \rangle, \langle o_1, \dots, o_m \rangle\rangle$  be an asynchronous Multi-Context System. A configuration of  $M$  is a pair*

$$CF = \langle\langle CF_1, \dots, CF_n \rangle, \langle OB_1, \dots, OB_m \rangle\rangle,$$

where

- for all  $1 \leq i \leq n$ ,  $CF_i = \langle kb, \mathbf{acc}, \mathbf{IB}, \mathbf{CM} \rangle$  is a configuration for  $C_i$  and for every output rule  $r \in OR_{CM}$  we have  $\mathbf{n} \in N(M)$  for  $\langle \mathbf{n}, \mathbf{i} \rangle = \text{head}(r)$ , and
- $OB_j = \dots, D_{l-1}, D_l$  is an information buffer with a final element  $D_l$  that corresponds to the data on the output stream named  $o_j$  for each  $1 \leq j \leq m$  such that for each  $h \leq l$  with  $D_h = \langle \mathbf{n}, \mathbf{i} \rangle$  we have  $\mathbf{n} = \mathbf{n}_{C_i}$  for some  $1 \leq i \leq n$ .

Similar to the context configuration, we have now a configuration of an asynchronous Multi-Context System. Intuitively it consists of one configuration for each context and a set of output buffers. In addition it is ensured that each output rule is referring to some context or output buffer and we do formalise that one output buffer is in fact just an input buffer which is not attached to a context.

These definitions now lead to the concept of a *run structure*, which will represent the changes over time in the dynamic environment. Note that we do not define how much time is between two different points in time and that we do not state any more assumptions towards this concept.

**Definition 6.1.9** (Run Structure). *Let  $M = \langle\langle C_1, \dots, C_n \rangle, \langle o_1, \dots, o_m \rangle\rangle$  be an asynchronous Multi-Context System. A run structure for  $M$  is a sequence*

$$R = \dots, CF^t, CF^{t+1}, CF^{t+2}, \dots,$$

where  $t \in \mathbb{Z}$  is a point in time, and every  $CF^t$  in  $R$  ( $t \in \mathbb{Z}$ ) is a configuration of  $M$ .

Because we want to take into account that computation takes time and that belief sets might need some time to be completely enumerated or put out, it is not enough to have such a notion of run structure. To model this computation, together with concurrent generation of answers and verification of non-existence of further belief sets, we will additionally introduce the boolean variable *busy* for each configuration in a run structure. Hence, context  $C_i$  is busy at time  $t$  if and only if *busy* is true for this context in this time instant. While one context is *busy* it will not process input stream data further, till all answers of the current computation are processed. As soon as one belief set is computed, the output rules determine which information is passed on to the different stakeholders (i.e. other contexts or output buffers). We will represent this by the notion of a *stakeholder buffer*. An information buffer  $B$  is the stakeholder buffer of  $C_i$  (for  $n$ ) at time  $t$  if

- $B = IB_{i'}^t$ , for some  $1 \leq i' \leq n$  such that  $n = n_{C_{i'}}$  is a stakeholder of some output rule in  $OR_{CM_{i'}^t}$  or
- $B = OB_{j'}^t$ , for some  $1 \leq j' \leq m$  such that  $n = o_{j'}$  is stakeholder of some output rule in  $OR_{CM_{i'}^t}$ .

Another newly used concept is that we want to provide a way to notify the contexts stakeholders about the end of a computation. To achieve this, we introduce the symbol  $\text{EOC} \in IL$  that will be sent to other contexts stakeholders as a notification that no more computation will be done<sup>38</sup>. Together with all these concepts, we will now define how an asynchronous Multi-Context System should behave during a run.

**Definition 6.1.10 (Run).** *Let  $M$  be an asynchronous Multi-Context System of length  $n$  with  $m$  output streams and  $R$  a run structure for  $M$ .  $R$  is a run for  $M$  if the following conditions hold for every  $1 \leq i \leq n$  and every  $1 \leq j \leq m$ :*

- (i) *if  $CF_i^t$  and  $CF_i^{t+1}$  are defined,  $C_i$  is neither busy nor waiting at time  $t$ , then*
- $C_i$  is busy at time  $t + 1$ ,
  - $CF_i^{t+1} = \mathbf{cu}_{CM_i^t}(IB_i^t, kb_i^t)$

*We say that  $C_i$  started a computation for  $kb_i^{t+1}$  at time  $t + 1$ .*

- (ii) *if  $C_i$  started a computation for  $kb$  at time  $t$  then*

- *we say that this computation ended at time  $t'$ , if  $t'$  is the earliest time point with  $t' \geq t$  such that  $\langle n_{C_i}, \text{EOC} \rangle$  is added to every stakeholder buffer of  $C_i$  at  $t'$ ; the addition of  $d_{C_i}(B, OR_{CM_i^{t'}}, n)$  to the buffer is called an end of computation notification.*

<sup>38</sup>Note that if some stakeholder only gets this EOC-token, it can deduce that either no output rule fired for the context or that that it could be verified that no belief set exists at all

- for all  $t' > t$  such that  $\text{CF}_i^{t'}$  is defined,  $C_i$  is busy at  $t'$  unless the computation ended at some time  $t''$  with  $t < t'' < t'$ .
  - if the computation ended at time  $t'$  and  $\text{CF}_i^{t'+1}$  is defined then  $C_i$  is not busy at  $t' + 1$ .
- (iii) if  $C_i$  started a computation for  $kb$  at time  $t$  that ended at time  $t'$  then for every belief set  $B \in \mathbf{acc}_i^t$  there is some time  $t''$  with  $t \leq t'' \leq t'$  such that
- $d_{C_i}(B, \text{OR}_{\text{CM}_i^{t''}}, \mathbf{n})$  is added to every stakeholder buffer of  $C_i$  for  $\mathbf{n}$  at  $t''$ .

We say that  $C_i$  computed  $B$  at time  $t''$ . The addition of the output  $d_{C_i}(B, \text{OR}_{\text{CM}_i^{t''}}, \mathbf{n})$  to some buffer is called a belief set notification.

- (iv) if  $\text{OB}_j^t$  and  $\text{OB}_j^{t+1}$  are defined and  $\text{OB}_j^t = \dots, D_{l-1}, D_l$  then  $\text{OB}_j^{t+1} = \dots, D_{l-1}, D_l, \dots, D_{l'}$  for some  $l' \geq l$ . Moreover, every data package  $D_{l''}$  with  $l < l'' \leq l'$  that was added at time  $t + 1$  results from an end of computation notification or a belief set notification.
- (v) if  $\text{CF}_i^t$  and  $\text{CF}_i^{t+1}$  are defined,  $C_i$  is busy or waiting at time  $t$ , and  $\text{IB}_i^t = D_1, \dots, D_l$  then we have  $\text{IB}_i^{t+1} = D_1, \dots, D_l, \dots, D_{l'}$  for some  $l' \geq l$ . Moreover, every data package  $D_{l''}$  with  $l < l'' \leq l'$  that was added at time  $t + 1$  either results from an end of computation notification or a belief set notification or  $\mathbf{n} \notin N(M)$  (i.e.  $\mathbf{n}$  is a sensor name) for  $D_{l''} = \langle \mathbf{n}, i \rangle$ .

Condition (i) describes how the transition from the idle phase to an ongoing computation is handled. How such a computation ends is formalised in (ii). There the behaviour about sending the EOC-token is defined too. In Condition (iii) it is stated that stakeholders are notified with the corresponding piece of information as soon as the belief set is computed. Additionally that means it might take variable amounts of time in between information deliveries about the same computation if it has more than one belief set. Finally, the addition of data towards an output stream or an input stream are expressed in Condition (iv) and (v) respectively. The transition of sensor data into input streams is implicit, which means that this information is just added from the outside and the system is modelling this mechanics any further. That implies that data from one sensor may appear at any time on some input streams and the only trace to the sensor is by its name appearing in the data package.



In summary the run is characterised as follows: Whenever a context  $C$  is not busy, its context controller  $\mathbf{cc}$  is computed to check whether a new computation should be done. If that is the case, the current configuration of the context is replaced by the newly generated one of the context update function  $\mathbf{cu}$  of  $C$ . This newly generated configuration needs to ensure that the newly generated input buffer is a suffix of the old one and a new computation for the updated knowledge base and semantics will start. After some undefined

span of time, belief sets are computed and based on the output rules of  $C$ , data packages are sent to stakeholder buffers. Then at some point of time, all stakeholders are notified by the context that the computation is finished, which is done with the EOC-token. After this notification the context is no longer busy.

Based on these definitions, it is obvious to see that many decisions have been deliberately abstract, like the undefined spans of time, the high dynamics of changing configurations, and the adaptive change of semantics. Nevertheless, we think that this is an easy to use formalism to model communication efforts between parts of other systems. In addition it is possible to express communication needed for stronger coupled semantics, like the one used for reactive Multi-Context Systems and might help to find bottlenecks for actual implementations and a common language ground to design and develop different parts of big interconnected systems.

## 6.2 Modelling an Example

Based on the suggestion in the previous section, which have proposed asynchronous Multi-Context Systems as a way to define communication paths and ways to formalise flow of information, we will now take a look on the second example scenario from Chapter 2. Our example deals with a recommender-system for the coordination and handling of ambulance assignments. The suggested asynchronous Multi-Context System supports decisions in various stages of an emergency case. It gives assistance during the rescue call, helps in assigning priorities and rescue units to a case, and assists in the necessary communication among all involved parties. The suggestions given by the system are based on different specialised systems which react to sensor readings. Moreover, the system can tolerate and incorporate overriding solutions proposed by the user that it considers non-optimal. We will try to show the intuition behind the system, therefore we will not go into detail for every context and system<sup>39</sup>.

Figure 6.1 depicts the example asynchronous Multi-Context System which models such a *Computer-Aided Emergency Team Management System* (CAET Management System). It uses  for sensor input,  for input streams. Green full arrows represent output rule related communication, while blue dotted arrows depict data flow from sensors. Note that interaction with a human (e.g. ER employee) is modelled as a pair containing an input stream and an output stream. The system consists of the following contexts:

**Case Analyser (ca)** This context implements a computer-aided call handling system which assists an emergency response employee (ER employee) dur-

<sup>39</sup>This choice is deliberately done to show how such a system can be described on different levels of detail in real life. Again, this stresses the fact that we want to use this framework as a language for specifications and not as a formal system which competes on one system to be used as a solution to different reasoning problems

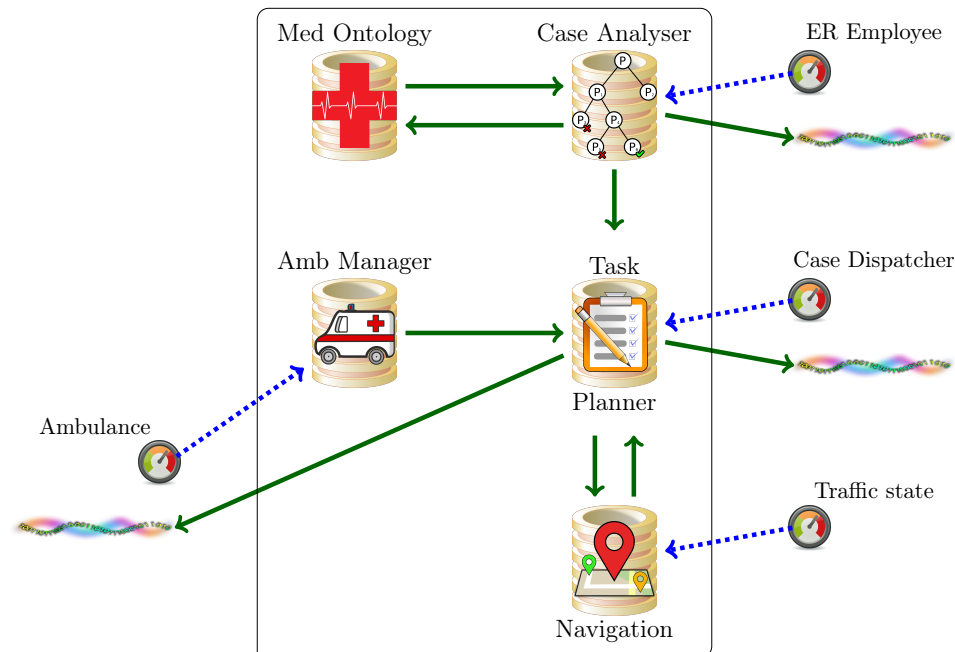


Figure 6.1: The Computer-Aided Emergency Team Management asynchronous Multi-Context System

ing answering an emergency call. The system utilises reasoning methods to choose which questions need to be asked based on previous answers. In addition, it may check whether answers are inconsistent (e.g. amniotic sac bursts when the gender is male). For these purposes the case analyser context may also consult a medical ontology represented by another context. The communication with the ER employee is represented, on the one hand, as a sensor that reads the input of the employee and, on the other hand, by an output stream which prints the questions and results on a computer screen.

To present the dialogue between the computer and the employee the system might use some kind of argumentation representation as additional graphical support for the human. The computed acceptable sets of arguments can then be used to communicate further on. Output rules would then use argument names as atoms to decide which information shall be forwarded to stakeholders and which should not.

During the collection of all the important facts for this emergency case, the analyser computes the priority of the case and passes it to the task planner.

**Med Ontology (mo)** This medical ontology can be realised, e.g. by a description logic reasoner which handles requests from the case analyser and



returns more specific knowledge about ongoing cases. This information may be used for the prioritisation of the importance of a case.

**Task Planner (tp)** This context keeps track of emergency cases. Based on the priority and age of a case and the availability and position of ambulances it suggests an efficient plan of action for the ambulances to the (human) case dispatcher (cd). The dispatcher may approve some of the suggestions or all of them. If the dispatcher has no faith in the given plan of action, she can also alter it at will. These decisions are reported back to the planning system such that it can react to the alterations and provide further suggestions. Based on the final plan, the task planner informs the ambulance about their new mission.

The knowledge base of the context is an answer-set program for reasoning about a suggested plan. It gets the availability and position of the ambulances by the ambulance manager. In addition, the cases with their priority are provided by the case analyser. With this information, the task planner gives the locations of the ambulances together with the target locations of the cases to a navigation system which provides the distances (i.e. the estimated time of arrival (ETA)) of all the ambulances to all the locations.

**Amb Manager (amb)** The ambulance manager is a database, which keeps track of the status and location of ambulance units. Each ambulance team reports its status (e.g. to be on duty, waiting for new mission, ...) to the database (modelled by the sensor “Ambulance” (amb)). Additionally, the car periodically sends GPS-coordinates to the database. These updates will be pushed to the task planner.

**Navigation (na)** This part of the asynchronous Multi-Context System gets traffic information (e.g. congestions, roadblocks, construction zones, ...) to predict the travel time for each route as accurate as possible. The task planner may push a query to the navigation system, which consists of a list of locations of ambulance units and a list of locations of target areas. Based on all the given information this context will return a ranking for each target area, representing the ETAs for each ambulance.

Now we want to have a closer look on the instantiation details of some aspects of our example. At first we investigate the  $\mathcal{CC}$  function of the case analyser. It allows for the computation of new belief sets whenever the ER employee pushes new information to the analyser. In addition, it will also approve of a new computation if the medical ontology supplies some requested information. Recall that the case analyser also assigns a priority to each case and that we want to allow the employee to set the priority manually. Let us suppose that such a manual override occurs and that the case analyser has an ongoing query to the medical ontology. Due to the manual priority assignment, the

requested information from the ontology is no longer needed. Therefore, it would be desirable that  $\mathcal{CC}$  does not allow for a recomputation if all conclusions of the ontology are only related to the manually prioritised case. With the same argumentation in mind, the context update function **cu** will ignore this information on the input stream too. This kind of behaviour may need knowledge about past queries which can be provided by an additional output rule for the case analyser which feeds the relevant information back to the context.

Next, we will have a look at the task planner that is based on answer-set programming. We will only present parts of the program, to show how the mechanics are intended to work. To represent the incoming information on the input stream, the following predicates can be used:

**case(caseid,loc,priority)** represents an active case (with its location and priority) which needs to be assigned to an ambulance.

**avail(amb,loc)** states the location of an available ambulance.

**eta(caseid,amb,value)** provides the estimated time of arrival for a unit at the location of the target area of the case.

**assign(amb,caseid)** represents the assignment of an ambulance to a case by the dispatcher.

These predicates will be added by the context update function to the knowledge base if corresponding information is put on the input stream of the context. Based on this knowledge, the other components of the answer-set program will compute the belief sets (e.g. via the stable model semantics). Note that an already assigned ambulance or case will not be handled as an available ambulance or an active case, respectively. In addition, **cu** can (and should) also manage forgetting of no longer needed knowledge. For our scenario it may be suitable to remove all **eta**, **avail** and **case** predicates when the cases or the unit is assigned. The **assign** predicate can be removed when the ambulance manager reports that the assigned ambulance is available again.

The set  $OR$  of output rules of the task planner could contain the following rules:<sup>40</sup>

$$\begin{aligned} \langle \text{cd,assign}(A,C) \rangle &\leftarrow \text{sugassignment}(A,C) \\ \langle \text{na,queryA}(L) \rangle &\leftarrow \text{avail}(A), \text{not assign}(A, \_), \text{loc}(A,L) \\ \langle \text{na,queryC}(L) \rangle &\leftarrow \text{case}(C,P), \text{loc}(A,L), \text{not assign}(A, \_) \\ \langle \text{amb,assigned}(A,C) \rangle &\leftarrow \text{assign}(A,C) \end{aligned}$$

The first rule informs the case dispatcher (cd) about a suggested assignment that has been computed by the answer-set program. Rules two and three

---

<sup>40</sup>Keep in mind that in an actual implementation one may want to provide further information via communication.

prepare lists of ambulances and cases for querying the navigation context. Recall that the latter needs a list of ambulance locations (generated by rule two) and a list of target area locations (generated by rule three). Also keep in mind that for each belief set a data package with all information for one context or output stream is constructed. So the whole list of current target areas and free ambulance units will be passed to the navigation context at once. The last rule notifies the ambulance team that it has been assigned to a specific case.

Related to this example we want to mention privacy aspects as a real world policy which is especially important to applications in public services and health care. As the multi-context system is a heterogeneous system with different contexts, a completely free exchange of data may be against privacy policies. This issue can be addressed by the adequate design of output rules, which can also be altered with respect to additional information in the input stream (e.g. some context gains the permission to receive real names instead of anonymous data). So each context may decide by its own which parts of the belief sets are shared and exchanged with other contexts.

Another interesting aspect about asynchronous Multi-Context Systems is the possibility to easily join two asynchronous Multi-Context Systems together, outsource a subset of contexts in a new asynchronous Multi-Context System, or to view an asynchronous Multi-Context System as an abstract context for another asynchronous Multi-Context System in a modular way. This can be achieved due to the abstract communication by means of streams. With respect to our scenario there could be some asynchronous Multi-Context System which does the management of resources for hospitals (e.g. free beds with their capabilities). The task planner might communicate with this system to take the needed services for a case into account (e.g. intensive care unit) and informs the hospital via these streams about incoming patients. It would be easy to join both asynchronous Multi-Context Systems together to one big system or to outsource some contexts as input sensors paired with an output stream. In addition, one may also combine different contexts or a whole asynchronous Multi-Context System to one abstract context to provide a dynamic granularity of information about the system and to group different reasoning tasks together.

### 6.3 Simulating Reactive Multi-Context Systems

To show that asynchronous Multi-Context Systems are expressive enough to handle the complexities of a reactive Multi-Context System, we will now show how to simulate an arbitrary reactive Multi-Context System by means of an asynchronous Multi-Context System. Of course one might just define one context which computes the equilibria and therefore be done, but that would not be very meaningful. Therefore we will only use context formalisms which are on par with the reactive Multi-Context System in question. Note that a

simulation in the other direction (i.e. simulating an asynchronous Multi-Context System by means of a reactive one) is not feasible, as the synchronised nature of the equilibrium would not allow concurrent computation of information. In addition it would not be possible to deliver results just as they emerge for the same reasoning. Another difference is that each context can utilise the computation controller and the context update function to choose whether one or all answers of other contexts are relevant for computation.

Due to this big set of differences, asynchronous Multi-Context Systems have another source of non-determinism. As computation time, communication time, and deliberate decision-making for input stream conversion and start of computations are completely asynchronous but still allowed to be different each time, it is not possible to present a deterministic outcome in a general setting<sup>41</sup>.

The simulation of a reactive Multi-Context System will also demonstrate that asynchronous Multi-Context Systems are a viable language frame to describe other formalisms with respect to communication and computation efforts. In other words, the asynchronous approach on the reactive system will give insight into possible implementation details for such a system. Additionally we get an example where some (in this case all) contexts get synchronised by means of a *management context* which controls the flow of computation between different contexts and tells them via specific input stream information that a computation needs to be started.

Lets assume we have some arbitrary reactive Multi-Context System

$$M_r = \langle C_r, IL_r, BR_r \rangle$$

with  $n$  different contexts  $\langle C_1, \dots, C_n \rangle$  and sets of bridge rules  $\langle BR_1, \dots, BR_n \rangle$ , as well as  $k$  different input languages for streams  $\langle IL_1, \dots, IL_k \rangle$ , we will construct an asynchronous Multi-Context System  $M_a$  with one output stream. To simulate the behaviour of each context  $C_i \in C_r$ , we introduce for every  $C_i$  the following contexts:

- a context  $C_{kb_i}$  that stores the current knowledge base of context  $C_i$  and will only change if the knowledge base is updated due to an equilibrium and an updated knowledge base due to **next**-operator changes,
- a context  $C_{acc_i}$  which accepts a knowledge base candidate and computes its semantics, and
- a context  $C_{m_i}$  that implements the bridge rules and the management function of the context.

---

<sup>41</sup>Although, due to thoughtful modelling of the contexts and their management methods, control is possible, as shown by the simulation of a reactive Multi-Context System.

In addition we will need some global contexts, which control the flow of information and enforce that the semantics of reactive Multi-Context Systems are followed:

- $C_{in}$  is receiving all the input for  $M_r$  and distributes these information to every context  $C_{m_i}$  where at least one bridge rule is dependant on the given input language information. It is keeping track of the current computation status and will only send information to the other contexts after the **next** operation has been finished.
- $C_g$  guesses equilibrium candidates for  $M_r$  and passes them to the management contexts  $C_{m_i}$ .
- $C_c$  compares every belief set it gets from  $C_{acc_i}$  and the one guessed by  $C_g$ . In case it is not matching, it notifies  $C_g$  to guess a new candidate. If an equilibrium is found, then the management contexts  $C_{m_i}$  are informed to apply the **next** operator bridge rules and send their updated knowledge base to their respective  $C_{kb_i}$ . In addition the input context  $C_{in}$  is informed to distribute the new input and  $C_g$  gets informed that it needs to start a new round of guessing equilibria candidates. Finally the context will write the equilibrium to the output stream to inform the outside world about the result of the computation.

## 6.4 Declarative Data Set Packing

In this final section about asynchronous Multi-Context Systems, we will now take a further look into the principles of input streams for contexts. In general, when dealing with stream data, it is always important to keep track of the information buffered in the input stream. Data might be already outdated, not completely represented by the pieces in the current stream, or just not relevant for the current state of computation. The basic idea of *declarative data set packing* is to combine the incoming data into packages, which should either be kept or directly used for computation, while dismissing other not relevant data<sup>42</sup>. This can be seen as some kind of preemptive filter on incoming information, before the actual computation controller and the context update function is triggered. Note that this filter can be seen as some part of a combined computation controller and context update function which makes a rough approximation to make it easier for them to process the input buffer data. As this kind of task can be seen as a combination of classification and configuration task, we think that answer set programming is an easy to utilise and powerful formalism to use. Answer set programming has in many cases the advantage that small changes in the requirements will imply small changes in the problem

<sup>42</sup>Note that we are calling the packages of information data set and not data package, as it can be some information provided by an input stream as well as by a context output rule.

encoding for solving the described problem. In addition configuration problems have been solved efficiently with answer set programming tools too [Soininen et al., 2001, Gebser et al., 2011d]. This is why we use an answer set programming point of view on this data packaging. In other words, we will use some answer set programming encodings and the deduced predicates from the stable models of the program will control on how to deal with the data from the input buffer.

The basic intuitive idea about the packaging is that we do not inspect the content of the data, but do reasoning on behalf of the meta-data around the package. This meta data can either be produced during the packaging step or on the other hand it can be provided by the sending context to make it easier for its stakeholders to parse the given information. Some of this information is already implicit in the system, like that a computation can have more than one data package and that each computation is finished by sending the EOC-token. In this approach we will extend this notion and give every computation a unique identifier, so it is clear whether one data package belongs to a current computation or is just some delayed piece of information from previous computations. To give additional information about some set of data can be done by tagging the information. These tags are just normal ground information associated with one or more data packages generated by output rules. It might be some information about a given computation (e.g. the current optimum value of an optimisation task or priorities of some given information). An important aspect on this concept is, that it can be added to the current concept of asynchronous Multi-Context Systems per context and that this filter system can be used in a very modular and optional way.

Table 6.1 provides an overview on the different built-in atoms which can be used to define the data packaging. The associated *type* reflects whether these atoms are expected to be part of the input provided in the input buffer or as a directive. Note that the usage of other atoms and predicates is allowed, but only these defined directives will cause some action from the context on the input buffer. We have chosen to have two different variants of computations. The first variant packs one so-called *package schema* per answer set, while the second one allows for multiple schemata per answer set. We are aware that the kind of lists used for the second variant is not supported in answer set programming in a native way. Still it is possible to nest binary and unary predicates to simulate a list<sup>43</sup>. This allows us to treat packs from different schemata differently with only one computation of an answer set. In the following we will discuss some of the basic design choices and options we need to take into account when using data set packing.

**Triggering Computation** When and how often such a computation shall be done to filter the input buffer depends highly on the needs of the application.

---

<sup>43</sup>In the following sections we will use an answer set programming style notation of `#list{elements}` to represent some language feature to construct these lists

<i>Atom</i>	<i>Type</i>	<i>Description</i>
<code>dsAvail(ds)</code>	Input	Data set <code>ds</code> is available on the input buffer
<code>dsComp(ds, comp)</code>	Input	Data set <code>ds</code> belongs to computation <code>comp</code>
<code>source(comp, ctxt)</code>	Input	Computation <code>comp</code> is a computation of context <code>ctxt</code>
<code>source(ds, ctxt)</code>	Input	Data set <code>ds</code> originates from context <code>ctxt</code>
<code>source(ds, therm)</code>	Input	Data set <code>ds</code> originates from sensor <code>therm</code>
<code>eoc(comp)</code>	Input	Computations <code>comp</code> has ended.
<code>tag(comp, solves(prob1))</code>	Input	Computations <code>comp</code> is tagged with function <code>solves(prob1)</code>
<code>tag(ds, "optimum")</code>	Input	Data set <code>ds</code> is tagged with string "optimum"
<code>time(1000)</code>	Input	An external clock provides 1000 as current time
<code>ignore(comp)</code>	Directive	Ignore future data sets of computation <code>comp</code>
<code>addTag(comp, best(3))</code>	Directive	Tag computation <code>comp</code> with function <code>best(3)</code>
<code>rmTag(comp, "trusted")</code>	Directive	Remove the tag "trusted" from computation <code>comp</code>
<code>rm(comp)</code>	Directive	Remove all data sets of computation <code>comp</code> from the input buffer
<code>rm(ds)</code>	Directive	Remove data set <code>ds</code> from the input buffer
<code>rmPack</code>	Directive	Remove all data sets of processed packages from the input buffer
<code>rmPack(sch)</code>	Directive	Remove all data sets of processed packages in schema <code>sch</code> from the input buffer
Variant: one package schema per answer set		
<code>inPack(ds)</code>	Directive	Data set <code>ds</code> is considered part of the package
<code>process(sch)</code>	Directive	The data sets defined by <code>in_pack/1</code> atoms form a package of schema <code>sch</code> and are passed on for processing
Variant: multiple package schemata per answer set		
<code>process(sch, [ds1, ds3, ds7])</code>	Directive	The data sets in the list <code>[ds1, ds3, ds7]</code> form a package of schema <code>sch</code> and are passed on for processing

Table 6.1: Input and directive atoms for package specifications. Atoms of other predicates can be used as needed and are considered auxiliary.

It might make sense to re-evaluate the conclusions every time new data arrives on the input stream if the evaluation is fast enough or can be done in an iterative way, and if it is imperative that the context reacts as fast as possible to new information. Sometimes it is sufficient to wait till a computation is completely done and the EOC-token is received and in other instances some interval-based check is sufficient. We will not decide which version should be used and therefore let it open to be decided on a per context basis.

**Number of Answer Sets** As we are executing the directives based on the answer sets of the corresponding encoding, we need to keep in mind that there might be multiple stable models in place. There are different ways on how to

handle this fact and how to utilise this kind of feature. In general we can see every computed answer set equally valid, therefore the easiest way of handling that issue is to only allow the asynchronous Multi-Context System-engine to compute one answer set and process the input buffer based on the concluded directives. It might sound very ignorant to dismiss the other results, but as there is no order of answer sets and every one is grounded as well as sound with respect to the rules, there is a good and comprehensible reason for each answer set to be one. Of course it is still possible to compute multiple answer sets and apply the directives on every answer set. Another possible solution might be to introduce optimisation to find the optimal set of directives for the filtering. The last option is to design the answer set program in such a way that all different directives are pooled together in one answer set. Again we want to let the modeller of the system decide which way is the best for the application. In addition the choice of the used option will depend on how the computation is triggered too.

To allow these different ways of implementation, we do provide two different kinds of directives to do the schema packing. In the first one, we only state that some data set is part of a schema package and we then just decide that this one should be processed. The second one allows to define different schemata per answer set, where different data sets might be part of one or more schemata.

In the following examples we will show how input and programs might look for different contexts. There we will use the computer aided emergency team management system, which we have introduced in Section 6.2.

**Example 6.4.1.** *As seen in the introduction of the task planner, we will get information about the availability of ambulances from the ambulance manager context. The case analyser context is providing information about the current cases. We assume that each data from the case analyser represents exactly one case, which has been prioritised and categorised already. A possible input buffer for the task planner might contain the following facts:*

```
dsAvail(ca_ds11). dsAvail(ca_ds12).  
dsAvail(amb_ds54). dsAvail(amb_ds55). dsAvail(amb_ds56).  
source(ca_ds11,ctxt_case_anl). source(ca_ds12,ctxt_case_anl).  
source(amb_ds54,ctxt_amb_mng). source(amb_ds55,ctxt_amb_mng).  
source(amb_ds56,ctxt_amb_mng).
```

*Each data set which is available has a unique name and one of the two contexts as a source. Note that this is just another representation of the information which is already sent by means of the output relevant for one stakeholder (see Definition 6.1.6). To pack the incoming information together, we might use an*



*answer set program, like*

```

auxCaseAvail ← dsAvail(X), source(X, ctxt_case_anl).
auxAmbAvail ← dsAvail(X), source(X, ctxt_amb_mng).
process(sch1) ← auxCaseAvail.
process(sch2) ← auxAmbAvail, not auxCaseAvail.
inPack(X) ← dsAvail(X), source(X, ctxt_amb_mng).
inPack(X) ← dsAvail(X), source(X, ctxt_case_anl).
rmPack ← .

```

The first two rules use auxiliary predicates to project that some information is available from the case analyser or the ambulance manager respectively. Then some package schema *sch1* is produced if at least one new data on a case from the case analyser is available. If no new case data is available, but some updates on the ambulances, the schema *sch2* is instead packaged. Both schemata will pack ambulance manager and case analyser data into a package, but *sch1* will only be created if only a new case is available. This kind of meta information can be used for different handling of the two cases and due to the semantics of answer set programs it is still ensured that only one answer set is computed. Rules 5 and 6 are then collecting the data sets into the package and finally the directive from the last rule will remove all the packaged data sets from the input buffer. Note that the third rule which produces the schema *sch1* will also do some processing, which means that it is safe to remove the packaged raw data from the input buffer.

This example has shown how to use the basic information which is already implicitly available by asynchronous Multi-Context Systems. In the next one we will go further and include reasoning about computations.

**Example 6.4.2.** *Before we have assumed that the case analyser computes only one belief set. We will now assume that it is instead trying to find better case classifications by doing some kind of optimisation process. The current best data package will be sent, but it will only be ensured to be an optimal one if the EOC-token is received too. This can be represented by the following input:*

```

dsAvail(ca_ds21). dsAvail(ca_ds22). dsAvail(ca_ds23). dsAvail(ca_ds25).
dsAvail(amb_ds54). dsAvail(amb_ds55). dsAvail(amb_ds56).
dsComp(ca_ds21, comp9). dsComp(ca_ds22, comp9). dsComp(ca_ds23, comp9).
dsComp(ca_ds25, comp10).
eoc(comp9).
source(ca_ds21, ctxt_case_anl). source(ca_ds22, ctxt_case_anl).
source(ca_ds23, ctxt_case_anl). source(ca_ds25, ctxt_case_anl).
source(am_ds54, ctxt_amb_mng). source(am_ds55, ctxt_amb_mng).
source(am_ds56, ctxt_amb_mng).
source(comp9, ctxt_case_anl). source(comp10, ctxt_case_anl).

```

Here we can now see that the information from the case analyser is organised in different computations. Computation `comp9` has already finished, because the `EOC`-token is already represented. The other computation `comp10` is still running and has not come to a conclusion so far. Because we know that the case analyser uses an optimisation enumeration, we are only interested in the newest package from the finished computation. This can be expressed by the following set of rules:

$$\begin{aligned} \text{finishedComp}(X) &\leftarrow \text{eoc}(X). \\ \text{inPack}(X) &\leftarrow \#max\{X : \text{dsAvail}(X), \text{finishedComp}(Y), \\ &\quad \text{dsComp}(X, Y), \text{source}(Y, \text{ctxt\_case\_anl})\}. \\ \text{inPack}(X) &\leftarrow \text{dsAvail}(X), \text{source}(X, \text{ctxt\_amb\_anl}). \\ \text{process}(\text{sch}) &\leftarrow \text{inPack}(X). \\ \text{rmPack} &\leftarrow \text{process}(X). \end{aligned}$$

We have changed the rules a little bit up to show how this could work out in a simple and straightforward way. First we collect all the information about already finished computations and then in the second rule we pool only these packages together which have the maximal number in the computation (i.e. the one with the highest number and therefore the last one which got sent). In addition we are still collecting the information about ambulances on the fly. Then we process a schemata if at least one data set got packaged and if something got processed on, we will remove all processed packages.

Another feature we have not shown in examples so far is that some information might be tagged by either a context or by the answer set encoding. This tagging adds additional flat information on top of the given meta information for the different data sets getting fed into the input buffer.

**Example 6.4.3.** To see how tagging might be utilised we will have another look at the example we are developing so far. Normally it can happen that some of the cases are so severe that it would be dangerous to wait for an optimal classification of the case (e.g. in case of an heart attack every second counts). Therefore the case analyser might add a priority tag to the data set, to notify about a severe emergency. To represent this for computation `comp10`, we will just add the following fact to the set input: `tag(comp10, severe)`. To be able to react to this severe case in a special way we will now update the encoding to show how the program can handle multiple packaging schemata at once.

$$\begin{aligned}
finishedComp(X) &\leftarrow eoc(X). \\
inPack(X) &\leftarrow \#max\{X : dsAvail(X), finishedComp(Y), \\
&\quad dsComp(X, Y), source(Y, ctxt\_case\_anl)\}. \\
inPack(X) &\leftarrow dsAvail(X), source(X, ctxt\_amb\_anl). \\
process(sch, L) &\leftarrow inPack(X), L = \#list\{Y : inPack(Y)\}. \\
auxEmerg(C) &\leftarrow not eoc(C), tag(C, severe). \\
process(em, L) &\leftarrow L = \#list\{Y : dsAvail(Y), dsComp(Y, C), \\
&\quad auxEmerg(C)\}. \\
rmPack(sch) &\leftarrow process(sch, \_). \\
rm(X) &\leftarrow auxEmerg(C), dsAvail(X), dsComp(X, C).
\end{aligned}$$

The parts on packaging the finished computation remains the same, except that the packaged data sets are pooled into a list, to be still processed under the schemata *sch*. Based on the tag, a new auxiliary predicate is generated, if its computation is not already done. Then an emergency schemata *em* is packaged with a list of all the importantly tagged data sets. Note that in case that more than one data set of the computation is already arrived this solution would package both data sets into it. The last two rules will be responsible to clean up all the packaged data sets. It should be mentioned that the data sets for the severe computations are removed on the fly, which means that at the point when the computation is done, no more available data sets are there and it will not be packaged a second time.

Although we have presented asynchronous Multi-Context Systems as a formalism to describe communication between different other contexts and knowledge representation formalisms, we have chosen to introduce this data set packing. We think that these examples and the concept of declarative data set packing is again an argument for this view on asynchronous Multi-Context Systems. The problem of fast and easy to define filtering of stream data is a problem that occurs in almost every other stream reasoning related work. By using an asynchronous Multi-Context System to define this declarative method of information filtering we are allowing other researchers and developers to adapt this technique for their own systems and we could introduce a declarative and easy to adapt solution for this overall problem.



---

## Related Work

---

In this thesis we have introduced two systems which are handling the integration of different belief sets from heterogeneous contexts and are reacting dynamically to changes over time represented by streams. This overall idea and motivation is not new though, so there exist several systems which are modelling the dynamics of knowledge and the flow of information. We will present the most relevant approaches to reactive reasoning formalisms in this section and compare them with reactive Multi-Context Systems in particular. Of course there are other systems, like *abstract modular systems* [Lierler and Truszczyński, 2016], which present approaches to model communication between different reasoning modules. This system uses some joint vocabularies instead of bridge rules for the exchange of communication, but it is only providing a solution to the knowledge integration problem and not the dynamics over time. Therefore we will focus on these systems which allow reasoning over time and incorporation and exchange of information.

When presenting these alternatives to reactive Multi-Context Systems we will also show whether our system is capable to model the concepts of these other systems in a comparative matter. This will be done in some comparative study about the most prominent related formalisms. Note that this section is more focused towards reactive Multi-Context Systems than on asynchronous Multi-Context Systems, because we have already demonstrated that the asynchronous ones are more general and we still see them as a modelling language and not as an actual implemented framework.

Nevertheless, we do want to mention that asynchronous Multi-Context Systems have similarities in common with the concept of *Multi-Agent Systems* [Ferber, 1999] because these systems are interested in communication between different artificial intelligence based agents. In a nutshell the biggest difference between context and agent systems is that an agent has its own motivation and goals. Therefore concepts like trust, liability, and personal goals are very important key components in Multi-Agent Systems, while we are more interested in the underlying problem of modelling and integrating

the information between the contexts. It can be seen as a more basic and general approach to formalise the communication, integration of knowledge, and dynamic reaction to events before getting to the point where the different components start to compete for their own intrinsic goals and schemes.

Section 5.2.4 is dealing with inconsistency management on a global scale for reactive Multi-Context Systems. The concepts have already been studied for managed Multi-Context Systems [Weinzierl, 2014, Eiter et al., 2014] and are based on the seminal work on inconsistency handling by Reiter [Reiter, 1987]. A further extension of these concepts has only recently been introduced and discussed, where some measurements of inconsistency are used to find and classify inconsistencies [Ulbricht et al., 2018]. It is some interesting topic to check whether and how these measurements can be adapted and used for reactive Multi-Context Systems, which we will see as a promising direction of further work.

In the following we will discuss two recent approaches to implement stream reasoning, which are LARS [Beck et al., 2015] and STARQL [Özgep et al., 2013]. Then we will have a look on reactive logic programming, which has been proposed in the form of EVOLP [Alferes et al., 2002] and reactive answer set programming [Gebser et al., 2011a]. Finally we will discuss recently introduced Multi-Context Systems, like streaming Multi-Context Systems [Dao-Tran and Eiter, 2017] and timed Multi-Context Systems [Cabalar et al., 2017].

## 7.1 Stream Reasoning Approaches

The task of reasoning over stream data becomes a topic which gets an increased measure of interest. One reason for that is because concepts like the *Internet of Things* [Want et al., 2015, Atzori et al., 2010] got more prominent in the past years and the Internet is seen more and more as a provider of services than merely a collection of different websites.

Reactive Multi-Context Systems are a well-suited formalism to do stream reasoning, because they can address the important challenges which are arising in this environment. The most important feature is that the system can decide by itself whether some information should be kept for later usage or not, as it has been shown in Section 5.2.3. In addition we have shown that we can utilise the common practice of sliding windows, which are used for temporal reasoning, but are not limited to that concept of temporal methods of forgetting and allows more sophisticated methods<sup>44</sup> of decisions to forget.

### LARS

The formal framework *Logic-based framework for Analyzing Reasoning over Streams* (LARS) [Beck et al., 2015] provides a formal logic-based language to

---

<sup>44</sup>See [Lin and R.Reiter, 1994] for an overview of formal forgetting.

reason with streams over time windows. Basically it is utilising a rule-based concept to provide mechanisms to do reasoning over streams. By introducing a novel window operator working and allowing to use different levels of abstraction of time, this whole framework is an elegant way for flexible representation and reasoning on streaming data. LARS has its semantics defined for a fixed input data stream and is based on the FLP computation [Faber et al., 2004] for a fixed point in time. In detail, the basic LARS language is built over a set of atoms  $\mathcal{A}$  which are defined over disjoint sets of predicates  $\mathcal{P}$  and constants  $\mathcal{C}$ . In addition to this usual definition, the predicates are divided into two disjoint sets of *extensional predicates*  $\mathcal{P}^{\mathcal{E}}$  to represent input streams and *intensional predicates*  $\mathcal{P}^{\mathcal{I}}$  which are used for intermediate and output streams.

Given  $i, j \in \mathbb{N}$ , the set of the form  $[i, j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$  is called an *interval*. An *evaluation function* over an interval  $T$  is a function  $v : \mathbb{N} \rightarrow 2^{\mathcal{A}}$ , such that  $v(t) = \emptyset$  if  $t \notin T$ . Further on, LARS defines a *stream* as a tuple  $s = \langle T, v \rangle$  with  $T$  being an interval and  $v$  an evaluation function over  $T$ . If a stream only contains extensional predicates it is called a *data stream*. A stream  $S' = \langle T', v' \rangle$  is a *substream* of a stream  $S = \langle T, v \rangle$  if  $T' \subseteq T$  and  $\forall t \in T' : v(t) \subseteq v'(t)$  holds. This is denoted by  $S' \subseteq S$ . The syntax of LARS formulae is defined by the following grammar:

$$\alpha := a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \diamond\alpha \mid \square\alpha \mid @_t\alpha \mid \boxplus_{\iota}^{\mathbf{x}}\alpha$$

where  $a \in \mathcal{A}$  and  $t \in \mathbb{N}$ . For the connectives (i.e.  $\neg, \wedge, \vee$ , and  $\rightarrow$ ) the usual classical intention is used. The  $\diamond$  and  $\square$  operators are utilised in the same way as in modal logics, such that a formula holds for some or every time instant during some interval. To represent some specific formula at a given time instant, the *exact operator*  $@_t$  is introduced. The novel operator  $\boxplus_{\iota}^{\mathbf{x}}$  allows one to have a focus on recent substreams. Here  $\iota$  is representing the window type used for the substream and  $\mathbf{x}$  is a tuple of parameters to specify the range of the substream. Two examples of this time-based window operators presented in [Beck et al., 2015] are  $\boxplus_{\tau}^n$ , which allows to represent the substream of the last  $n$  time instants of a stream and  $\boxplus_p^{\text{id}\mathbf{x}, n}$  that allows one to first split the stream into substreams on basis of the evaluation function to later only refer to the last  $n$  time instants of that substream.

A LARS program is based on the concept of rules in a similar way as Answer Set Programming is utilising them. In detail, a LARS rule is of the form

$$\alpha \leftarrow \beta_1, \dots, \beta_j, \text{not } \beta_{j+1}, \dots, \text{not } \beta_n$$

where  $\alpha, \beta_1, \dots, \beta_n$  are formulae and  $\alpha$  consists only of intensional predicates.

The semantics of LARS are based on the FLP semantics of logic programs. For a LARS program  $P$  with a given input stream  $D = \langle T, v \rangle$  the *answer streams* of  $P$  for  $D$  are computed for each time instant  $t \in T$  and the associated substream of  $D$  with the static semantics of answer set programming techniques over the fixed windows which are implicitly given for each operator. Note that

the  $\diamond$  and  $\square$  operators can only be checked over the associated substream of  $D$ , which means that the window size for the substreams will dictate how long the history of previous stream data will be and how general (respectively abstract) the statements of these operators are.

For getting a better idea on how such a rule might look like, we will use a simplified version of the example used in [Beck et al., 2015]. There some reasoning over a network of trams in a public transport network is modelled with LARS.

$$\begin{aligned} @_{T}\text{exp}(Id, Y) \leftarrow \boxplus_p^{\text{idx},1}@_{T_1}\text{tram}(Id, X), \text{line}(Id, L), \text{not } \boxplus_\tau^{20} \diamond \text{jam}(X), \\ \text{plan}(L, X, Y, Z), T = T_1 + Z. \end{aligned}$$

The predicate **plan** represents that a line  $L$  has two consecutive stops  $X$  and  $Y$  with a planned travel time of  $Z$ , while **line** expresses that one specific tram train with an  $Id$  is operating on line  $L$ . **jam** is representing the information that some station  $X$  is jammed and the expression  $\text{not } \boxplus_\tau^{20} \diamond \text{jam}(X)$  formalises that during the last twenty time instants there should have been no frame where a jam has been reported. The last used expression in the body **tram** is stating that the tram train  $Id$  is at station  $X$ .  $\boxplus_p^{\text{idx},1}@_{T_1}\text{tram}(Id, X)$  utilises the window operator to focus only on the last report of  $\text{tram}(Id, X)$  and  $@_{T_1}$  is formulating that this event should have been at time  $T_1$ . On an intuitive level the whole rule states that the tram  $Id$  can be expected to be at station  $Y$  at time  $T$  if the tram has been on the previous station  $X$  at  $Z$  minutes of the estimated travel time ago and no jam has been reported for the station  $X$  in the last 20 minutes.

Now we want to show that a reactive Multi-Context System can be constructed which is simulating such a LARS reasoning process in a native way.

More precisely, we assume fixed sets of predicates  $\mathcal{P}$  and constants  $\mathcal{C}$ , a fixed window size  $w \in \mathbb{N}$  and a LARS program  $P$  over  $\mathcal{A}$ , the set of atoms obtained from  $\mathcal{P}$  and  $\mathcal{C}$ . Let  $\mathcal{A}^{\mathcal{E}}$  be the set of atoms that include only extensional predicates from  $\mathcal{P}^{\mathcal{E}}$ , and  $\mathcal{A}^T$  be the set of time-tagged atoms, i.e.,  $\mathcal{A}^T = \{\langle a, t \rangle \mid a \in \mathcal{A} \text{ and } t \in \mathbb{N}\}$ .

Consider  $M = \langle \langle C_{\text{LARS}}, \langle IL_1, \text{Clock} \rangle, \langle BR_{C_{\text{LARS}}} \rangle \rangle \rangle$  as a reactive Multi-Context-System obtained from  $P$  and  $w$  in the following way:

- $C_{\text{LARS}} = \langle L, OP, \mathbf{mng} \rangle$  where
- $L = \langle KB, BS, \mathbf{acc} \rangle$  is such that
- $KB = \{P \cup A \cup \{\text{now}(t)\} \mid A \subseteq \mathcal{A}^T \text{ and } t \in \mathbb{N}\}$
- $BS = \{S \mid S \text{ is a stream for } \mathcal{A}\}$
- $\mathbf{acc}(kb) = \{S \mid S \text{ is an answer stream of } P \text{ for } D^{kb} \text{ at time } t^{kb}\}$ 
  - $t^{kb} = t$  such that  $\text{now}(t) \in kb$
  - $T^{kb} = [t^{kb} - w, t^{kb}]$



- $v^{kb}(t) = \{a \mid \langle a, t \rangle \in kb\}$ , for each  $t \in \mathbb{N}$
- $D^{kb} = \langle T^{kb}, v^{kb} \rangle$
- $OP = \{\text{add}(\delta) \mid \delta \in \mathcal{A}^T\} \cup \{\text{del}(\delta) \mid \delta \in \mathcal{A}^T\} \cup \{\text{add}(\text{now}(t)) \mid t \in \mathbb{N}\}$
- $\text{mng}(kb, op) = (kb \cup \{\delta \mid \text{add}(\delta) \in op\}) \setminus \{\delta \mid \text{del}(\delta) \in op\}$
- $IL_1 = \mathcal{A}^\mathcal{E}$
- $Clock = \mathbb{N}$
- $BR_{C_{\text{LARS}}}$  contains the following rules for managing history:

$$\begin{aligned}
&\text{add}(\text{now}(T)) \leftarrow \text{clock}::T \\
&\text{add}(\langle A, T \rangle) \leftarrow 1::A, \text{clock}::T \\
&\text{del}(\langle A, T \rangle) \leftarrow \text{clock}::T', T < T' - w \\
&\text{next}(\text{add}(\langle A, T \rangle)) \leftarrow 1::A, \text{clock}::T \\
&\text{next}(\text{del}(\langle A, T \rangle)) \leftarrow \text{clock}::T', T \leq T' - w
\end{aligned}$$

Given an input stream  $\mathcal{I}$  for  $M$  and a time point  $t \in \mathbb{N}$ , we consider  $t_t^\mathcal{I}$ , the unique element of stream  $Clock$  at step  $t$ , which represents the current time at step  $t$ . We also consider the LARS input data stream at time  $t$ ,  $D_t^\mathcal{I} = \langle T, v \rangle$ , such that  $T = [t_t^\mathcal{I} - w, t_t^\mathcal{I}]$  and  $v(t') = \{a \in \mathcal{A}^\mathcal{E} \mid \text{there exists } t'' \leq t \text{ such that } t' = t_{t''}^\mathcal{I} \text{ and } a \in \mathcal{I}_1^{t''}\}$  for  $t' \in T$ , and  $v(t') = \emptyset$  otherwise. Then, given an input stream  $\mathcal{I}$  for  $M$ , at each time point  $t \in \mathbb{N}$ , each equilibria stream  $\mathcal{B}$  for  $M$  given  $\text{KB} = \langle \{P\} \rangle$  and  $\mathcal{I}$  is composed of an answer stream of  $P$  for  $D_t^\mathcal{I}$  at time  $t_t^\mathcal{I}$ .

Note that at each time instant the knowledge base contains only the relevant part of the (possibly infinite) input stream, meaning that information no longer valid is really discarded, and that the current time, given by the stream  $Clock$ , is decoupled from the time steps at which equilibria are evaluated. For the sake of an easier to read presentation, we have assumed a fixed time window  $w$ , yet an extension in the spirit of what we presented in Section 5.2.3 on forgetting can easily be used to allow dynamic windows.

## STARQL

To represent data from different sources in an useful way, the *Resource Description Framework*(RDF)-Standard [W3C RDF Working Group, 2014] is maintained by the *World Wide Web Consortium*. A prominent language to answer queries on these structured data is the SPARQL *Protocol and RDF Query Language*(SPARQL<sup>45</sup>) [W3C SPARQL Working Group, 2013]. An extension of this concept is *continuous* SPARQL (C-SPARQL) [Barbieri et al., 2010], which allows to do queries over time. Alas, this language is only focusing on answering queries in a very basic way. The *Optique Project* [Rodríguez-Muro

<sup>45</sup>SPARQL is a recursive acronym.

et al., 2013] has aimed on adding intelligent reasoning on top of this query language and proposed the *Streaming and Temporal ontology Access with a Reasoning-based Query Language* (STARQL <sup>46</sup>) STARQL [Özçep et al., 2013] which allows reasoning on base of Description Logic assertions.

In more detail, every stream utilised in the STARQL framework consists of timestamped  $\mathcal{A}\text{Box}$  assertions, which will be used for input streams, as well as output streams which are considered the answers of the queries. A query on such a stream is structured in the following way:

```
SELECT selectClause( $\vec{x}, \vec{y}$ )
FROM listOfWindowedStreamExpressions
USING listOfResources
WHERE  $\Psi(\vec{x})$ 
SEQUENCE BY seqMethod
HAVING  $\Phi(\vec{x}, \vec{y})$ 
```

We would invite the interested reader for extensive descriptions of semantics and syntax of STARQL to read the Optique Deliverable 5.1 [Özçep et al., 2013]. For a better intuition on the core aspects as well as the usage of this formalism we will take one of the examples used by the authors of the Deliverable document to explain the components of STARQL queries.

We assume that there is a  $\mathcal{T}\text{Box}$  located at <http://example.org/TBox><sup>47</sup>, which contains

$$\textit{BurnerTipTempSensor} \sqsubseteq \textit{TempSensor}$$

as an axiom. That axiom is stating that every Burnertip temperature sensor is a temperature sensor too. An additional  $\mathcal{A}\text{Box}$  is located at <http://example.org/staticABox>, which defines that some sensor  $s_0$  is a Burnertip temperature sensor by  $\textit{BurnerTipSensor}(s_0)$ . We consider the following input stream of sensor readings:

$$S = \{rd(s_0, 90)\langle 0s \rangle \\ rd(s_1, 30)\langle 0s \rangle \\ rd(s_0, 93)\langle 1s \rangle \\ rd(s_1, 32)\langle 1s \rangle \\ rd(s_0, 94)\langle 2s \rangle \\ rd(s_0, 91)\langle 3s \rangle \\ rd(s_0, 93)\langle 4s \rangle \\ rd(s_0, 95)\langle 5s \rangle\}$$

The stream  $S$  contains time-tagged information from two sensors  $s_0$  and  $s_1$ . Note that the ontology only defines  $s_0$  as a sensor and does not provide any

---

<sup>46</sup>pronounced Star-Q-L

<sup>47</sup>Note that these are just example domains. This file does not exist there, but the STARQL-standard makes it mandatory that resources are located at specific URIs.

classification on  $s_1$ . A valid STARQL-query on this ontology and the given stream could be as follows:

```
CREATE STREAM S_out AS

SELECT {?sens rdf:type MonIncTemp}<NOW>
FROM S 0s<-[NOW-2s, NOW]->1s
USING  STATIC ABOX <http://example.org/staticABox>,
        TBOX <http://example.org/TBox>
WHERE { ?sens rdf:type TempSensor }
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i<= j in SEQ1 ,x,y:
        IF ( { ?sens rd ?x }<i> AND { ?sens rd ?y }<j> )
        THEN ?x <= ?y
```

In words, this query will have a stream  $S_{out}$  as its output. This stream will consist of assertions of the type *MonIncTemp*, which states that the temperature of sensor readings has increased monotone. The FROM statement now defines that the data for the reasoning should be taken from the stream  $S$  and utilising a sliding window, such that the current time instant and the preceding two seconds will be used. It is specifying the width of steps too, i.e. that the query should be evaluated every second. Afterwards the static  $\mathcal{A}Box$  and the utilised  $\mathcal{T}Box$  is specified. The line with the WHERE command is then filtering the input stream to only analyse sensor data which is classified as a sensor (i.e. only sensor  $s_0$  in the current example). The last two statements are then forming a sequence structure over the remaining stream data and use the built-in arithmetical functions to describe the concept of monotone increasing sensor readings. Note that during the query different variables get bound via the WHERE, SEQUENCE BY, and HAVING parts explicitly, while binding, naming, and classification based on the ontology is implicitly done.

In a more detailed manner, we will now have a look on how the data is partitioned step by step by this query. First the input stream gets aligned to the current time points into a so-called *temporal ABox*. How the different time-stamped axioms are gathered into the corresponding window is shown in Table 7.1. When the data is filtered towards the description logic classification and reorganised into a sequence, the data is represented as a sequence of timestamped  $\mathcal{A}Boxes$ . In an example where more sensor data per timestamp would be still available it would be easy to see that all these pooled information is gathered into one  $\mathcal{A}Box$ , which is tagged with the associated timestamp. Our example has the windowed stream

$$\{\{rd(s_0, 93)\}\langle 1s \rangle, \{rd(s_0, 94)\}\langle 2s \rangle, \{rd(s_0, 91)\}\langle 3s \rangle\}$$

at time point 3 after applying the sequencing method, and the classification from the description logic reasoner. If we run the STARQL-query as intended at every second, it will produce the following output stream  $S_{out}$ :

Time	Temporal ABox
0s	$\{rd(s_0, 90)\langle 0s \rangle, rd(s_1, 30)\langle 0s \rangle\}$
1s	$\{rd(s_0, 90)\langle 0s \rangle, rd(s_1, 30)\langle 0s \rangle, rd(s_0, 93)\langle 1s \rangle, rd(s_1, 32)\langle 1s \rangle\}$
2s	$\{rd(s_0, 90)\langle 0s \rangle, rd(s_1, 30)\langle 0s \rangle, rd(s_0, 93)\langle 1s \rangle, rd(s_1, 32)\langle 1s \rangle, rd(s_0, 94)\langle 2s \rangle\}$
3s	$\{rd(s_0, 93)\langle 1s \rangle, rd(s_1, 32)\langle 1s \rangle, rd(s_0, 94)\langle 2s \rangle, rd(s_0, 91)\langle 3s \rangle\}$
4s	$\{rd(s_0, 94)\langle 2s \rangle, rd(s_0, 91)\langle 3s \rangle, rd(s_0, 93)\langle 4s \rangle\}$
5s	$\{rd(s_0, 91)\langle 3s \rangle, rd(s_0, 93)\langle 4s \rangle, rd(s_0, 95)\langle 5s \rangle\}$

Table 7.1: Temporal ABox of the input stream  $S$ 

$$\begin{aligned}
S\_out = \{ & MonIncTemp(s_0)\langle 0s \rangle \\
& MonIncTemp(s_0)\langle 1s \rangle \\
& MonIncTemp(s_0)\langle 2s \rangle \\
& MonIncTemp(s_0)\langle 5s \rangle \}
\end{aligned}$$

We will now present one possible way to design a reactive Multi-Context System which is capable of realising STARQL queries as introduced just beforehand. Note that this illustration will be more on a meta-level than the one previously shown for LARS, because the standard of STARQL has many different options and ways to manipulate the queries, which can be realised on different ways due to the abstract and general style of reactive Multi-Context Systems. First, we assume that one computed equilibrium correlates directly to one STARQL-query evaluation for one time point. In addition we will design this reactive Multi-Context System with an external clock in mind to keep track of the current time. The given input stream is realised as a sensor input which gets the sensor readings directly at the current time instant. Therefore the incoming sensor information does not need to be timestamped any more in this setup and we will omit this information in our input stream. Note that it is easy to add the timestamps, but that would also allow to send sensor data for other timestamps, which might not be intended for the current example and needs to be decided per application. We see this big amount of possible ways to implement such a system as a feature, because it can be decided by the modeller which variant is the best fitting for the given problem.

Figure 7.1 illustrates which contexts we are assuming to implement a STARQL-query. The input stream  $\mathcal{I}_S$  is representing the stream data on which the query should be evaluated and  $\mathcal{I}_c$  is just a clock to inform the reactive Multi-Context System about the current point in time. Their information is packed together in a simple storage context  $C_{tA}$  to represent the temporal ABox of the input stream (as pictured in Table 7.1). Further on that context is keeping track of the current window size, such that it deletes already obsolete data

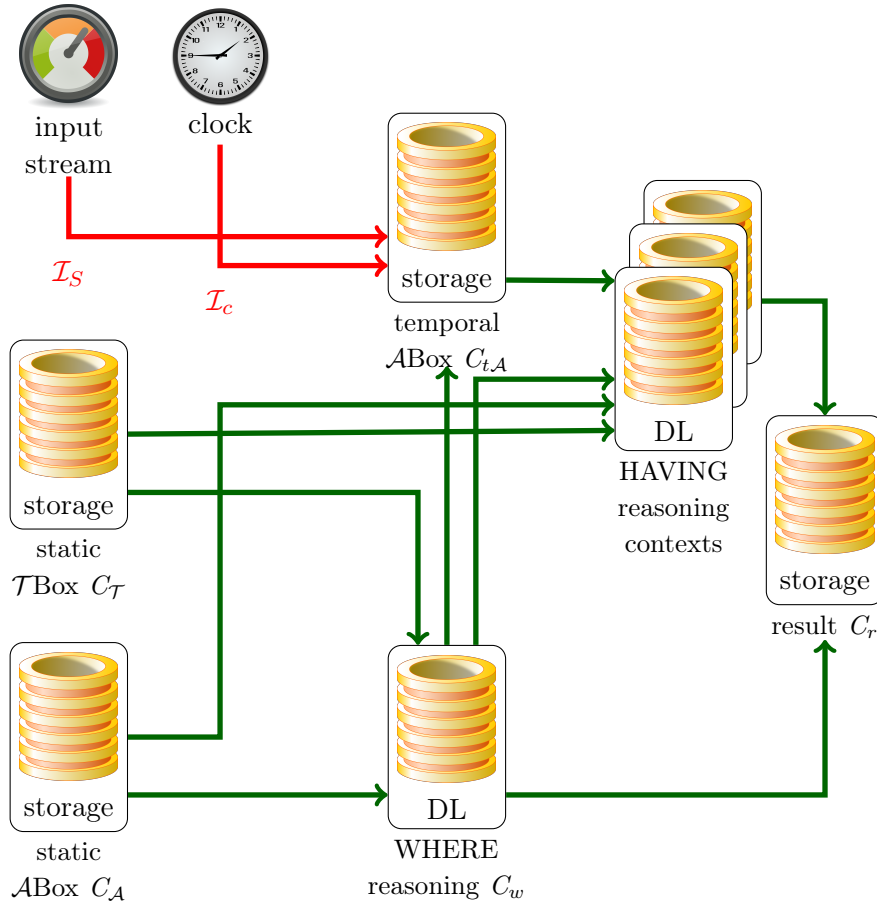


Figure 7.1: A reactive Multi-Context System to capture the semantics of a STARQL-query-engine

from its knowledge base. In addition cut out data (i.e. the data which is not related due to the WHERE part of the query) which is not used and therefore not visible from the original stream. The classification and variable binding, which stems from the WHERE part is handled by the context  $C_w$ . It gets the ontology from the two contexts  $C_T$  and  $C_A$  and delivers the variable bindings to the other contexts which need to have knowledge about the variables used in the query. The imported assertions and axioms from these two description logic contexts which is defined via the bridge rules of  $C_w$  are derived from the USING directive of the query and implement the whole semantics of it by the different bridge rules. Note that these information is imported into the contexts responsible for the HAVING directive too. To implement it, we are assuming a set of contexts, utilising description logic to compute the HAVING semantics per one piece of the sequence given by the SEQUENCE BY statement. That means we are assuming one context per sequence instance. In our example we

have a window size of three, so the operator `stdSeq` can only generate up to three sequences. Therefore we show three different contexts in the figure. Each reasoning context will evaluate one time instant of the variable. To evaluate the for all expression in the HAVING part, we utilise the last context  $C_r$  which evaluates all the results of the HAVING contexts with respect to the variable bindings, provided by  $C_w$  to compute the result of the query at the current time. The knowledge base of  $C_w$  will represent the results of the query at the given point in time.

## 7.2 Reactive Logic Programming

### EVOLP

Evolving logic programs (EVOLP [Alferes et al., 2002]) are an extension of logic programs, to allow a program to *evolve* over time, such that it can alter its rules over a sequence of computations. In addition they can handle external information in the form of *events*, which are seen as a sequence of other evolving logic programs that are added to represent additional external input. Intuitively EVOLP utilises logic programs as discussed in Section 4.2, with the addition that default negation is allowed in the head of the rules. Such a program can be seen as a program over the language  $\mathcal{L}$ , which is extended by a set of assertion. Assertions are noted by an unary atom `assert`. These assertions may be atoms and rules as well, which allows nesting of rules in the assertion extension.  $\mathcal{L}$  combined with the assertions creates an extended language  $\mathcal{L}_{\text{assert}}$  which is formally inductively defined by the following steps:

- (i) All propositional atoms in  $\mathcal{L}$  are atoms in  $\mathcal{L}_{\text{assert}}$ ;
- (ii) If  $R$  is a rule over  $\mathcal{L}_{\text{assert}}$ , then  $\text{assert}(R)$  is an atom in  $\mathcal{L}_{\text{assert}}$ ;
- (iii) nothing else is considered to be an atom in  $\mathcal{L}_{\text{assert}}$ .

EVOLP is a logic program with default negation in its head over  $\mathcal{L}_{\text{assert}}$ . To denote the set of all rules over  $\mathcal{L}_{\text{assert}}$ , we will write  $\mathcal{R}_{\mathcal{L}}$ . The basic idea behind these evolving logic programs is that they change over time such that the program starts with its basic program over the language  $\mathcal{L}$ . If some assertion `assert(R)` is considered true by the evaluation of the program, the rule  $R$  has to be considered as a rule in the next evaluation of the program. In addition the newly introduced rule is considered more important than the other rules, such that older rules will only be used if they do not conflict to the newer ones, which is the idea behind the similar concept of dynamic logic programs [Alferes et al., 2000].

The semantics of this concept is based on sequences of interpretations for the changing logic program. Formally, an *evolution interpretation* of length  $m$  of an evolving logic program  $P$  over a propositional language  $\mathcal{L}$  is a finite sequence

$I = \langle I^1, \dots, I^m \rangle$  of sets of propositional atoms of  $\mathcal{L}_{\text{assert}}$ . The evolution trace associated with an evolution interpretation  $I$  is the sequence of programs  $P_I = \langle P^1, \dots, P^m \rangle$  such that  $P^1 = P$  and, for each  $2 \leq j \leq m$ ,  $P^j = \{r \mid \text{assert}(r) \in I^{j-1}\}$ .

To allow reaction to external events, which are in the form of external rules which are going to be considered in further computations, we need to show how they are related too. In a more formal matter, given an evolving logic program  $P$  over  $\mathcal{L}$ , a sequence of evolving logic programs over  $\mathcal{L}$  is called an *event sequence over  $P$* . Let  $P$  be an evolving logic program over  $\mathcal{L}$ ,  $I = \langle I^1, \dots, I^m \rangle$  an evolution interpretation of length  $m$  of  $P$  with evolution trace  $P_I = \langle P^1, \dots, P^m \rangle$ , and  $E = \langle E^1, \dots, E^\ell \rangle$  an event sequence over  $P$  such that  $\ell \geq m$ . Then,  $I$  is an *evolution stable model* of  $P$  given  $E$  iff for every  $1 \leq j \leq m$ , we have that  $I^j$  is a stable model of  $P^1 \oplus P^2 \oplus \dots \oplus (P^j \cup E^j)$ . EVOLP calls this the notion of a central model of evolving logic programs.

It is easy to see that the idea of assertions is close towards the concept of the **next**-Operator, because both allow for changes on the knowledge base, based on the previous computation step. Another similarity is the integration of external knowledge, which is directly modelled for reactive Multi-Context Systems by the bridge rules. The biggest difference is that EVOLP is only considering answer set programming as the formalism to do the reasoning, while reactive Multi-Context System allows an arbitrary heterogeneous environment of context formalisms and more kinds of manipulation of the knowledge base through a more general management function.

To show this, we are going to present a reactive Multi-Context System  $M_P$ , which will simulate the behaviour of a EVOLP-system. The system  $M_P = \langle \langle C \rangle, \langle IL \rangle, \langle BR_C \rangle \rangle$  contains exactly one context  $C = \langle L, OP, \mathbf{mng} \rangle$ , where

- $L = \langle KB, BS, \mathbf{acc} \rangle$  is a logic such that
  - $KB$  is the set of pairs  $\langle D, E \rangle$  where  $D$  is a dynamic logic program over  $\mathcal{L}$ , and  $E$  is an evolving logic program over  $\mathcal{L}$
  - $BS$  is the set of all subsets of  $\mathcal{L}_{\text{assert}}$
  - $\mathbf{acc}(\langle P^1 \oplus \dots \oplus P^j, E \rangle)$  is the set of stable models of  $P^1 \oplus \dots \oplus P^{j-1} \oplus (P^j \cup E)$
- $OP = \{\text{as}(r) \mid r \in \mathcal{R}_{\mathcal{L}}\} \cup \{\text{ob}(r) \mid r \in \mathcal{R}_{\mathcal{L}}\}$
- $\mathbf{mng}(op, \langle D, E \rangle) = \{\langle D \oplus U, E' \rangle\}$  where  $U = \{r \in \mathcal{R}_{\mathcal{L}} \mid \text{as}(r) \in op\}$  and  $E' = \{r \in \mathcal{R}_{\mathcal{L}} \mid \text{ob}(r) \in op\}$ ,

and the input stream language is defined as  $IL = \mathcal{R}_{\mathcal{L}}$ . The used bridge rules are defined such that

$$BR_C = \{\mathbf{next}(\text{as}(s)) \leftarrow 1:\text{assert}(s) \mid \text{assert}(s) \in \mathcal{L}_{\text{assert}}\} \cup \{\text{ob}(s) \leftarrow 1::s \mid s \in IL\}$$

hold. The intuitive idea behind that context is that it takes via an input stream the currently applied events. In the knowledge base, the logic consists of a pair, where one element is the current evolution trace of the evolving logic program and the second one is the currently applied set of event rules. Its semantics is then just the semantics of the dynamic logic program, constructed by the trace and the current program combined with the events. Note that because the events are only added for the equilibria computation, we do not need to remove them after they have been in effect and the **next**-operator based bridge rule will update the evolution trace with respect to the computed equilibrium.

This leads to a direct correspondence between the equilibria stream and the models for EVOLP: Given an event sequence  $E = \langle E_1, \dots, E_\ell \rangle$  over  $P$  consider its associated input stream  $\mathcal{I}_E = \langle \langle E_1 \rangle, \dots, \langle E_\ell \rangle \rangle$ . Then,  $I = \langle I^1, \dots, I^\ell \rangle$  is a evolution stable model of  $P$  given  $E$  iff  $I$  is an equilibria stream for  $M_P$  given  $\text{KB} = \langle \langle P, \emptyset \rangle \rangle$  and  $\mathcal{I}_E$ . This is no big surprise, because the conception of assert and the **next**-operator are very similar, and the context formalism of the reactive Multi-Context System is just utilising a slightly extended version of a dynamic logic program solver.

## Reactive Answer Set Programming

Other closely related reactive logic programming frameworks are the implemented solver `oclingo` [Gebser et al., 2012a] and one theoretically described in [Brewka, 2013]. The `oclingo`-system extends the answer set programming solver such that it can handle external modules which are provided during runtime by a controller. These external modules are producing output which is fed into the reasoner for the reactive program in similar ways than it is done in EVOLP. The biggest difference is that only atoms are allowed to be communicated. On the other hand `oclingo` allows to decide on the number of enumerated answer sets and therefore allows better tuning of the results and the solving time than EVOLP. Because reactive Multi-Context Systems allow a simulation of EVOLP and the reactive addition of `oclingo` only uses atoms, which is a subset of whole rules, it is not complicated to adapt the given simulation to work for `oclingo` too. In addition we can use either different ways to define the semantics of the context logic, as well (as shown in Section 5.2.3) as making adjustments due to bridge rules to the reasoning mode of the semantics. One big advantage on the side of `oclingo` is that it is an actual system, while for our presented systems an implementation can be considered as an interesting aspect of future work.

For the basic concept of reactive answer set programming from [Brewka, 2013], reactive Multi-Context System can capture that too, because the proposed evolutionary operators for the reactive answer set programming system can be captured by the operators of the reactive Multi-Context System.



## 7.3 Other Multi-Context Systems

Before we are discussing other Multi-Context Systems in the literature, I want to mention the joint work about reactive Multi-Context Systems [Brewka et al., 2018]. One section in this paper is considering another semantics, namely the well-founded semantics. Unlike the other presented parts, I had no share of work on this concept, this is why it was not presented in Section 5.2. The basic idea is to reduce the reactive Multi-Context System in such a way that a comparatively easy fix-point computation on an easy to compute operator can be used to approximate the result of the equilibria stream. It is not trivial to check for the property of being reducible and the reduction is only in few circumstances easily computed. Nevertheless in those cases where it is doable, the complexity for the  $Q^{\exists}$  is shown to be in  $\mathbf{P}$  if  $\mathcal{CC}(M, k:b)$  is in  $\mathbf{P}$  too.

Now we will investigate other recently introduced Multi-Context Systems with the aim to tackle the handling of dynamics over time.

### Streaming Multi-Context Systems

The concept of streaming Multi-Context Systems [Dao-Tran and Eiter, 2017] starts with the same idea as it has been for this work. They take the concept of managed Multi-Context Systems and extend it towards a data-flow environment for reactive reasoning. A very big conceptual difference between reactive Multi-Context Systems and streaming Multi-Context Systems is the focus of its point of view. While we have been focused on the integration of external information on the basis of stream data, which can be utilised in bridge rules, together with the concept of the **next**-operator to distinguish between declarative and operational manipulations, streaming Multi-Context Systems are more focused on the notion of reasoning over time. The design choice of using a deterministic management function as well as a logic instead of a logic suite, in contrast to the base contributions on Multi-Context Systems [Brewka and Eiter, 2007] and managed Multi-Context Systems [Brewka et al., 2011b], is similar for both stream reasoning frameworks.

Intuitively a streaming Multi-Context System has no language for integrating external sensor and stream data into their framework. Instead contexts are defined which just hold the information from such sensors as their knowledge base. Conception wise this is a very good idea with respect to the inconsistency management, because the omni-incoherence property, which has been mentioned in Section 5.2.4 is still holding with the right intention<sup>48</sup>. On the other hand, a way to model different trust mechanics on basis on the actual data and providing means for handling them in bridge rules enables one for more freedom and expressiveness at the level of the framework, without the need

---

<sup>48</sup>Because streaming Multi-Context Systems shift the stream handling into contexts, the knowledge and their conclusions may be seen as beliefs rather than constant facts as it is seen in reactive Multi-Context Systems

to manipulate other context formalisms. Note that the streaming approach is sticking to the concept of the old reactive Multi-Context Systems, where the knowledge bases used for the verification of the equilibrium are reused as the knowledge bases of the next computation step (i.e. the case where all declarative rules are duplicated to be operative rules too for reactive Multi-Context Systems). Another difference is, that the bridge rules in a streaming Multi-Context System are allowing LARS-expressions. That is showing the different focus of the work very well, because the system is very specialised on doing temporal reasoning and allowing the user to easily define assumptions and assertions on basis of sliding windows. Additionally this system is allowing to take computation time into account. Which means asynchronous computation. Similar to asynchronous Multi-Context System, they consider computation time, output time, as well as data transfer times into the formal model. The approach is pretty straightforward, such that each context is computing its local equilibrium on basis of all already known belief sets and using the old ones if another context is still computing. To some extent this approach is similar to asynchronous Multi-Context Systems, but we are considering methods for contexts to wait on each other and decide on itself (or by some management context) on when to do new computations (i.e. computation controller **cc**).

Streaming Multi-Context Systems are introducing the concept of *idealized runs*, where the contexts are bound to the steps again. In this setting it has been shown that streaming Multi-Context Systems can simulate the old reactive Multi-Context Systems and evolving Multi-Context Systems [Gonçalves et al., 2014] if the declarative bridge rules are the same as the operational ones. Therefore streaming Multi-Context Systems can still simulate reactive Multi-Context Systems with the idealized runs if the declarative and operational bridge rules are the same, but nevertheless they will suffer the same shortcomings which have been discussed on the old reactive Multi-Context Systems in Section 5.1. Compatibility wise we have shown that reactive Multi-Context Systems can simulate LARS, therefore it would be possible to simulate that behaviour in the bridge rules too. We see that as an advantage, because by that way we can control where the system needs to keep track of histories for the sliding windows and even use dynamic sliding windows (see Section 5.2.3 for examples and discussion).

In the bottom line all three systems, namely streaming Multi-Context Systems, reactive Multi-Context Systems, and asynchronous Multi-Context Systems have the same ideas in common, but are still different enough because they are focusing on different baseline problems. Additionally streaming Multi-Context Systems and reactive Multi-Context Systems are providing features which cannot be simulated by the other one directly.

### Timed Multi-Context Systems

The timed Multi-Context Systems [Cabalar et al., 2017] are a recently proposed draft on another take on the field of reasoning over time with Multi-Context Systems. A very big difference is, that external input is handled separated from the notion of an equilibrium. The system allows external information to be proposed to the system, and then the knowledge base is updated by a non-deterministic function. Based on this updated knowledge base, the management function is applied, as for declarative bridge rules and then an equilibrium is computed. In addition a timed Multi-Context System only allows totally coherent contexts with respect to the naming used by Definition 5.2.22. We would propose that it is a good sign that there are many different approaches which are very similar to the one we have already presented. This shows that the field has increased interest in the scientific community and that our presented concept has its merits.

From our point of view, there is one point of critique on this system, namely that the changes based on observations are not based on any reasoning or beliefs of the different contexts, but all the contexts have to adjust accordingly. In a general setting, where one needs to react to the outside world, this is a totally fine way of reasoning, but when taking thoughts about inconsistent sensor data, unreliable sensor information, and similar real word problems into account, we think that the idea to reason about this information within the framework of equilibria is more appropriate and foremost more adaptable than limiting one to applying these information beforehand and introducing possible inconsistencies into the knowledge base in a non-deterministic way. Note that in comparison with asynchronous Multi-Context System, we allow translation of the information via the context update function, which should handle and tackle such inconsistent events and even allows the function to adjust the output rules and other parts of the context management to adapt the context towards the incoming information.



---

# Conclusion

---

We have tackled two very prominent problems in the field of knowledge representation and reasoning in this work. This has been the integration of knowledge between different formalisms and the communication and exchange of beliefs between heterogeneous reasoners, and the handling of dynamics from environments where data flows potentially continuous over time in such a way that the framework can modify itself and the context knowledge to provide some stream of information which can be seen as a reaction to these environments.

One contribution is the overall introduction and discussion of reactive Multi-Context Systems, where we have presented and shown that the system can handle different problems which may arise in different real world scenarios. We have discussed on how to model different methods of reasoning and how to simulate different aspects we have changed compared to previous work (e.g. usage of a deterministic management function). In addition we provided some insight on how to maintain consistency during the computation of the equilibria stream by either restricting the context formalisms and bridge rules or by utilising repair techniques as well as partial equilibria semantics. For handling the already addressed knowledge integration part, like it has been done with managed Multi-Context Systems [Brewka et al., 2011b], we stood faithful with the already existing work and just added new methods and redefined most of the formalism to make it easier to use and to still make all the important properties of consistency and computational complexity applicable to the newly introduced work, which we consider as one of the main contributions; A formal step forward to allow integration and handling of dynamics, as a proper generalisation of the single computation formalism and enough expressiveness to control the flow of information between the contexts and over time.

In addition we have proposed another framework, which is even more general than reactive Multi-Context Systems, that allows for asynchronous communication between the different contexts and even more adaptation over time by changing its means to communicate over time. asynchronous Multi-Context Systems are also allowing for self-managed decisions about the right

time for computation and changes of the knowledge base on basis of the input. Due to the high abstraction level of this formalism we have chosen to present it more as a modelling language and framework to describe (and possibly unify) communication between intelligent reasoning systems on a data driven level.

Based on a variety of different newly emerged proposals for different Multi-Context Systems, as presented in Section 7.3, it is easy to see that the current field of knowledge representation and reasoning is interested in further advances into the direction of Multi-Context Systems, and due to a decent level of similarities between the systems the direction we have chosen and presented is justified very well. That the systems can capture different already existing solutions for more specialised applications, like window based stream reasoning and ontology based stream querying, is another good indication that our system is actually usable and well defined.

The frameworks introduced are highly abstract and it is needless to say that this has been on purpose. Without this intention it would not be possible to capture a broad range of situations and systems without imposing restrictions to the used knowledge representation formalisms. We have produced and illustrated the tool which allows the application to choose which formalism and modelling approach is the best suited for the task. Of course that makes it imperative that many things need to be instantiated at first hand before a running system is available. It begins with the choice of context formalisms and semantics, continues to the choice of operators and their actual implementation and order of importance in the manage function towards the design of the declarative and operational bridge rules. That holds for the dual case of asynchronous Multi-Context System too, where one needs to design the context management and needs to think about the whole communication protocol which is explicitly designed by the output rules, computation controller, and the context update function. But I still believe that these approaches deliver valuable and unique solutions to the problems outlined in the introduction and the motivation of this work.

### **Future Work**

One very important point of future work would be to see whether it is feasible to implement reactive Multi-Context Systems, especially because there is very much communication overhead, which could be seen in the simulation example at Section 6.3. This problem has already been encountered for [Dao-Tran et al., 2015], where they have investigated that the whole distributed guess and check semantics needs very much communication between the contexts. Another interesting implementation task would be to produce some black box testing related implementation for an asynchronous Multi-Context System, where one can check whether some implementation of a context does what is defined by the formal model. In addition to that some best practice packages would be

---

very helpful for developers, that they have some kind of library to get short implementations of reoccurring tasks.

Further future work would be an in-depth investigation of inconsistency measurement [Ulbricht et al., 2018] for further results to ensure consistency on computations. In addition some additional studies about applicability of the asynchronous Multi-Context System framework for already developed stream reasoning systems and big data applications would be interesting and would allow to additionally appeal to the commercial applications too.





---

# Bibliography

---

- [Alferes et al., 2000] Alferes, J., Leite, J., Pereira, L., Przymusinska, H., and Przymusinski, T. (2000). Dynamic updates of non-monotonic knowledge bases. *J. Log. Program.*, 45(1-3):43–70.
- [Alferes et al., 2002] Alferes, J. J., Brogi, A., Leite, J. A., and Pereira, L. M. (2002). Evolving logic programs. In Flesca, S., Greco, S., Leone, N., and Ianni, G., editors, *8th European Conference on Logics in Artificial Intelligence (JELIA 2002)*, volume 2424 of *Lecture Notes in Computer Science*, pages 50–61. Springer.
- [Anicic et al., 2012] Anicic, D., Rudolph, S., Fodor, P., and Stojanovic, N. (2012). Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407.
- [Arenas et al., 1999] Arenas, M., Bertossi, L. E., and Chomicki, J. (1999). Consistent query answers in inconsistent databases. In Vianu, V. and Papadimitriou, C. H., editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79. ACM Press.
- [Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.
- [Baader, 2003] Baader, F. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge University Press.
- [Barbieri et al., 2010] Barbieri, D., Braga, D., Ceri, S., Valle, E. D., and Grossniklaus, M. (2010). C-SPARQL: a continuous query language for RDF data streams. *Int. J. Semantic Computing*, 4(1):3–25.
- [Bauer et al., 2016] Bauer, H., Baur, C., Mohr, D., Tschiesner, A., Weskamp, T., Aliche, K., Mathis, R., Noterdaeme, O., Behrendt, A., Kelly, R., et al.

## BIBLIOGRAPHY

---

- (2016). Industry 4.0 after the initial hype—where manufacturers are finding value and how they can best capture it. *McKinsey Digital*.
- [Beck et al., 2015] Beck, H., Dao-Tran, M., Eiter, T., and Fink, M. (2015). LARS: A logic-based framework for analyzing reasoning over streams. In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1431–1438. AAAI Press.
- [Brewka, 2013] Brewka, G. (2013). Towards reactive multi-context systems. In Cabalar, P. and Son, T. C., editors, *12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, volume 8148 of *Lecture Notes in Computer Science*, pages 1–10, Corunna, Spain. Springer.
- [Brewka et al., 2017a] Brewka, G., Diller, M., Heissenberger, G., Linsbichler, T., and Woltran, S. (2017a). Solving advanced argumentation problems with answer-set programming. In Singh, S. P. and Markovitch, S., editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1077–1083. AAAI Press.
- [Brewka and Eiter, 2007] Brewka, G. and Eiter, T. (2007). Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pages 385–390. AAAI Press.
- [Brewka et al., 2011a] Brewka, G., Eiter, T., and Fink, M. (2011a). Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In Balduccini, M. and Son, T. C., editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, pages 233–258. Springer.
- [Brewka et al., 2011b] Brewka, G., Eiter, T., Fink, M., and Weinzierl, A. (2011b). Managed multi-context systems. In Walsh, T., editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 786–791. IJCAI/AAAI.
- [Brewka et al., 2011c] Brewka, G., Eiter, T., and Truszczyński, M. (2011c). Answer set programming at a glance. *Commun. ACM*, 54(12):92–103.
- [Brewka et al., 2016a] Brewka, G., Ellmauthaler, S., Gonçalves, R., Knorr, M., Leite, J., and Pührer, J. (2016a). Inconsistency management in reactive multi-context systems. In Michael, L. and Kakas, A. C., editors, *15th European Conference on Logics in Artificial Intelligence (JELIA 2016)*, volume 10021 of *Lecture Notes in Computer Science*, pages 529–535.

- [Brewka et al., 2016b] Brewka, G., Ellmauthaler, S., Gonçalves, R., Knorr, M., Leite, J., and Pührer, J. (2016b). Towards inconsistency management in reactive multi-context systems. In Booth, R., Casini, G., Klarman, S., Richard, G., and Varzincza, I. J., editors, *Proceedings of the International Workshop on Defeasible and Ampliative Reasoning (DARE-16) co-located with the 22th European Conference on Artificial Intelligence (ECAI 2016), The Hague, Holland, August 29, 2016.*, CEUR Workshop Proceedings. CEUR-WS.org.
- [Brewka et al., 2018] Brewka, G., Ellmauthaler, S., Gonçalves, R., Knorr, M., Leite, J., and Pührer, J. (2018). Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. *Artificial Intelligence*, 256:68–104.
- [Brewka et al., 2014a] Brewka, G., Ellmauthaler, S., and Pührer, J. (2014a). Multi-context systems for reactive reasoning in dynamic environments. In Ellmauthaler, S. and Pührer, J., editors, *International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014)*, pages 23–30.
- [Brewka et al., 2014b] Brewka, G., Ellmauthaler, S., and Pührer, J. (2014b). Multi-context systems for reactive reasoning in dynamic environments. In Schaub, T., Friedrich, G., and O’Sullivan, B., editors, *21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 159–164. IOS Press.
- [Brewka et al., 2013] Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J. P., and Woltran, S. (2013). Abstract dialectical frameworks revisited. In Rossi, F., editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. IJCAI/AAAI.
- [Brewka et al., 2017b] Brewka, G., Ellmauthaler, S., Strass, H., Wallner, J. P., and Woltran, S. (2017b). Abstract dialectical frameworks. an overview. *IfCoLog Journal of Logics and their Applications Volume 4, number 8. Formal Argumentation*, 4(8):2263–2317.
- [Brewka and Woltran, 2010] Brewka, G. and Woltran, S. (2010). Abstract dialectical frameworks. In Lin, F., Sattler, U., and Truszczyński, M., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference*, pages 102–111. AAAI Press.
- [Brewka and Woltran, 2014] Brewka, G. and Woltran, S. (2014). GRAPPA: A semantical framework for graph-based argument processing. In Schaub, T., Friedrich, G., and O’Sullivan, B., editors, *21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 153–158. IOS Press.

## BIBLIOGRAPHY

---

- [Cabalar et al., 2017] Cabalar, P., Costantini, S., and Formisano, A. (2017). Multi-context systems: Dynamics and evolution. In Bogaerts, B. and Harrison, A., editors, *Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms co-located with the 14th International Conference on Logic Programming and Nonmonotonic Reasoning, ASPOCP@LPNMR 2017, Espoo, Finland, July 3, 2017.*, volume 1868 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [Calimeri et al., 2015] Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., and Schaub, T. (2015). Asp-core-2: Input language format. *ASP Standardization Working Group, Tech. Rep.*
- [Caminada and Wu, 2011] Caminada, M. and Wu, Y. (2011). On the limitations of abstract argumentation. In Causmaecker, P. D., Maervoet, J., Messelis, T., Verbeeck, K., and Vermeulen, T., editors, *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*, pages 51–58.
- [Cheng et al., 2006] Cheng, F.-L., Eiter, T., Robinson, N., Sattar, A., and Wang, K. (2006). Lpforget: A system of forgetting in answer set programming. In Sattar, A. and Kang, B. H., editors, *19th Australian Joint Conference on Artificial Intelligence (AUSAI 2006)*, volume 4304 of *Lecture Notes in Computer Science*, pages 1101–1105. Springer.
- [Church, 1996] Church, A. (1996). *Introduction to Mathematical Logic, pt. 1 Reprint*. Princeton Univeristy Press.
- [Colmerauer et al., 1973] Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un systeme de communication homme-machine en francais. Technical report, Technical report, Groupe de Intelligence Artificielle Universite de Aix-Marseille II.
- [Cornet and de Keizer, 2008] Cornet, R. and de Keizer, N. (2008). Forty years of SNOMED: a literature review. *BMC Medical Informatics and Decision Making*, 8(Suppl 1):S2.1–6. Proceedings Supplement: Selected contributions to the First European Conference on SNOMED CT, Copenhagen, Denmark, Oct 1-3, 2006. Edited by Schulz, Stefan and Klein, Gunnar.
- [Dao-Tran and Eiter, 2017] Dao-Tran, M. and Eiter, T. (2017). Streaming multi-context systems. In Sierra, C., editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1000–1007. ijcai.org.
- [Dao-Tran et al., 2015] Dao-Tran, M., Eiter, T., Fink, M., and Krennwallner, T. (2015). Distributed evaluation of nonmonotonic multi-context systems. *J. Artif. Intell. Res.*, 52:543–600.

- [Denecker et al., 2000] Denecker, M., Marek, V. W., and Truszczyński, M. (2000). Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning. In *Logic-Based Artificial Intelligence*, pages 127–144. Kluwer Academic Publishers.
- [Denecker et al., 2004] Denecker, M., Marek, V. W., and Truszczyński, M. (2004). Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Inf. Comput.*, 192(1):84–121.
- [Diller et al., 2015] Diller, M., Wallner, J. P., and Woltran, S. (2015). Reasoning in abstract dialectical frameworks using quantified boolean formulas. *Argument & Computation*, 6(2):149–177.
- [Dung, 1995] Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358.
- [Eiter et al., 2014] Eiter, T., Fink, M., Schüller, P., and Weinzierl, A. (2014). Finding explanations of inconsistency in multi-context systems. *Artif. Intell.*, 216:233–274.
- [Eiter and Gottlob, 1995] Eiter, T. and Gottlob, G. (1995). On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323.
- [Ellmauthaler, 2012] Ellmauthaler, S. (2012). Abstract Dialectical Frameworks: Properties, Complexity, and Implementation. Master’s thesis, Technische Universität Wien, Institut für Informationssysteme.
- [Ellmauthaler, 2013] Ellmauthaler, S. (2013). Generalizing multi-context systems for reactive stream reasoning applications. In Jones, A. V. and Ng, N., editors, *Proceedings of the 2013 Imperial College Computing Student Workshop (ICCSW 2013)*, OpenAccess Series in Informatics (OASICs), pages 17–24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [Ellmauthaler and Pührer, 2014] Ellmauthaler, S. and Pührer, J. (2014). Asynchronous multi-context systems. In Ellmauthaler, S. and Pührer, J., editors, *International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014)*, pages 31–38.
- [Ellmauthaler and Pührer, 2015] Ellmauthaler, S. and Pührer, J. (2015). Asynchronous multi-context systems. In Eiter, T., Strass, H., Truszczyński, M., and Woltran, S., editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*. Springer.

## BIBLIOGRAPHY

---

- [Ellmauthaler and Pührer, 2016] Ellmauthaler, S. and Pührer, J. (2016). Stream packing for asynchronous multi-context systems using ASP. In Eiter, T., Faber, W., and Woltran, S., editors, *Proceedings of the Workshop on Trends and Applications of Answer Set Programming (TAASP 2016)*.
- [Ellmauthaler and Strass, 2013] Ellmauthaler, S. and Strass, H. (2013). The DIAMOND system for argumentation: Preliminary report. In Fink, M. and Lierler, Y., editors, *Proceedings of the 6th International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2013)*, volume abs/1312.6140.
- [Ellmauthaler and Strass, 2014] Ellmauthaler, S. and Strass, H. (2014). The DIAMOND system for computing with abstract dialectical frameworks. In Parsons, S., Oren, N., Reed, C., and Cerutti, F., editors, *5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 233–240. IOS Press.
- [Ellmauthaler and Strass, 2016] Ellmauthaler, S. and Strass, H. (2016). DIAMOND 3.0 - A native C++ implementation of DIAMOND. In *6th International Conference on Computational Models of Argument (COMMA 2016), Potsdam, Germany, 12-16 September, 2016.*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 471–472. IOS Press.
- [Ellmauthaler and Wallner, 2012] Ellmauthaler, S. and Wallner, J. P. (2012). Evaluating Abstract Dialectical Frameworks with ASP. In Verheij, B., Szeider, S., and Woltran, S., editors, *4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245, pages 505–506. IOS Press.
- [Faber et al., 2004] Faber, W., Leone, N., and Pfeifer, G. (2004). Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J. J. and Leite, J. A., editors, *9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 4, pages 200–212. Springer.
- [Ferber, 1999] Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.
- [Gabbay et al., 1993] Gabbay, D., Hogger, C., and Robinson, J. (1993). *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 1: Logic Foundations*. Handbook of Logic in Artificial Intelligence. Clarendon Press.
- [Gabbay and Guenther, 2014] Gabbay, D. M. and Guenther, F., editors (2001–2014). *Handbook of philosophical logic, Volume 1–17*. Springer.
- [Gaggl et al., 2015] Gaggl, S. A., Rudolph, S., and Strass, H. (2015). On the computational complexity of naive-based semantics for abstract dialectical frameworks. In Yang, Q. and Wooldridge, M., editors, *Proceedings of the*

- 
- 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2985–2991. IJCAI/AAAI.
- [Gebser et al., 2012a] Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., and Schaub, T. (2012a). Stream reasoning with answer set programming: Preliminary report. In Brewka, G., Eiter, T., and McIlraith, S. A., editors, *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI Press.
- [Gebser et al., 2011a] Gebser, M., Grote, T., Kaminski, R., and Schaub, T. (2011a). Reactive answer set programming. In Delgrande, J. P. and Faber, W., editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *LNCS*, pages 54–66. Springer.
- [Gebser et al., 2011b] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. (2011b). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124.
- [Gebser et al., 2012b] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012b). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers.
- [Gebser et al., 2011c] Gebser, M., Kaminski, R., König, A., and Schaub, T. (2011c). Advances in gringo series 3. *Logic programming and nonmonotonic reasoning*, pages 345–351.
- [Gebser et al., 2011d] Gebser, M., Kaminski, R., and Schaub, T. (2011d). aspcud: A linux package configuration tool based on answer set programming. In *Proceedings of the Second Workshop on Logics for Component Configuration (LoCoCo 2011)*, volume 65 of *EPTCS*, pages 12–25.
- [Gebser et al., 2011e] Gebser, M., Sabuncu, O., and Schaub, T. (2011e). An incremental answer set programming based system for finite model computation. *AI Commun.*, 24(2):195–212.
- [Gelfond and Kahl, 2014] Gelfond, M. and Kahl, Y. (2014). *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press.
- [Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080.
- [Gogic et al., 1995] Gogic, G., Kautz, H., Papadimitriou, C., and Selman, B. (1995). The comparative linguistics of knowledge representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 862–869. Morgan Kaufmann.

## BIBLIOGRAPHY

---

- [Gonçalves et al., 2014] Gonçalves, R., Knorr, M., and Leite, J. (2014). Evolving multi-context systems. In Schaub, T., Friedrich, G., and O’Sullivan, B., editors, *21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 375–380.
- [Heißenberger, 2016] Heißenberger, G. (2016). A system for advanced graphical argumentation formalisms. Master’s thesis, TU Wien. Available at <http://www.dbai.tuwien.ac.at/proj/adf/grappavis/>.
- [Hoehndorf et al., 2015] Hoehndorf, R., Schofield, P. N., and Gkoutos, G. V. (2015). The role of ontologies in biological and biomedical research: a functional perspective. *Briefings in Bioinformatics*, 16:1069.1–12.
- [Huth and Ryan, 2004] Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press.
- [Johnson, 1992] Johnson, D. S. (1992). *Handbook of Theoretical Computer Science: Algorithms and Complexity*, chapter 2: A Catalog of Complexity Classes, pages 68–162. MIT Press Cambridge.
- [Kowalski, 1988] Kowalski, R. A. (1988). The early years of logic programming. *Communications of the ACM*, 31(1):38–43.
- [Lang and Marquis, 2010] Lang, J. and Marquis, P. (2010). Reasoning under inconsistency: A forgetting-based approach. *Artif. Intell.*, 174(12-13):799–823.
- [Le-Phuoc et al., 2012] Le-Phuoc, D., Parreira, J. X., and Hauswirth, M. (2012). Linked stream data processing. In Eiter, T. and Krennwallner, T., editors, *8th International Summer School on Reasoning Web (RW 2012)*, volume 7487 of *Lecture Notes in Computer Science*, pages 245–289. Springer.
- [Lécué and Pan, 2013] Lécué, F. and Pan, J. Z. (2013). Predicting knowledge in an ontology stream. In Rossi, F., editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 2662–2669. IJCAI/AAAI.
- [Lierler and Truszczyński, 2016] Lierler, Y. and Truszczyński, M. (2016). On abstract modular inference systems and solvers. *Artificial Intelligence*, 236:65–89.
- [Lin and R.Reiter, 1994] Lin, F. and R.Reiter (1994). Forget it! In Greiner, R. and Subramanian, D., editors, *Working Notes of the AAAI Fall Symposium on Relevance*, pages 154–159.



- [Motik et al., 2009] Motik, B., Shearer, R., and Horrocks, I. (2009). Hyper-tableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228.
- [Niemelä and Simons, 1996] Niemelä, I. and Simons, P. (1996). Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSP)*, pages 289–303.
- [Özçep et al., 2013] Özçep, Özgür. L., Möller, R., Neuenstadt, C., Zheleznyakov, D., Kharlamov, E., Horrocks, I., Hubauer, T., and Roshchin, M. (2013). D5.1 Executive Summary: A semantics for temporal and stream-based query answering in an OBDA context. Technical report, Optique FP7-ICT-2011-8-318338 Project.
- [Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.
- [Polberg and Wallner, 2017] Polberg, S. and Wallner, J. P. (2017). Preliminary report on complexity analysis of extension-based semantics of abstract dialectical frameworks. Technical Report DBAI-TR-2017-103, TU Wien.
- [Redl, 2014] Redl, C. (2014). *Answer Set Programming with External Sources: Algorithms and Efficient Evaluation*. PhD thesis, Vienna University of Technology, Knowledge-Based Systems Group, A-1040 Vienna, Karlsplatz 13.
- [Reiter, 1987] Reiter, R. (1987). A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95.
- [Richardson and Ruby, 2008] Richardson, L. and Ruby, S. (2008). *RESTful web services*. " O'Reilly Media, Inc."
- [Rodriguez-Muro et al., 2013] Rodriguez-Muro, M., Jupp, S., and Srinivas, K., editors (2013). *The Optique Project: Towards OBDA Systems for Industry (Short Paper)*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- [Rothmaler, 2000] Rothmaler, P. (2000). *Introduction to Model Theory*. Algebra, Logic, and Applications. Gordon & Breach.
- [Salkin et al., 2018] Salkin, C., Oner, M., Ustundag, A., and Cevikcan, E. (2018). A conceptual framework for industry 4.0. In *Industry 4.0: Managing The Digital Transformation*, pages 3–23. Springer.
- [Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53.

## BIBLIOGRAPHY

---

- [Soininen et al., 2001] Soininen, T., Niemelä, I., Tiihonen, J., and Sulonen, R. (2001). Representing configuration knowledge with weight constraint rules. In *Proceedings of the First International Workshop on Answer Set Programming (ASP 2001), Towards Efficient and Scalable Knowledge Representation and Reasoning*.
- [Strass, 2013] Strass, H. (2013). Approximating Operators and Semantics for Abstract Dialectical Frameworks. *Artificial Intelligence*, 205:39–70.
- [Strass, 2015a] Strass, H. (2015a). Expressiveness of two-valued semantics for abstract dialectical frameworks. *J. Artif. Intell. Res. (JAIR)*, 54:193–231.
- [Strass, 2015b] Strass, H. (2015b). The relative expressiveness of abstract argumentation and logic programming. In Koenig, S. and Bonet, B., editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1625–1631. AAAI Press.
- [Strass and Ellmauthaler, 2017] Strass, H. and Ellmauthaler, S. (2017). go-DIAMOND 0.6.6 – ICCMA 2017 system description. Second International Competition on Computational Models of Argumentation: <http://www.dbai.tuwien.ac.at/iccma17/>.
- [Strass and Wallner, 2015] Strass, H. and Wallner, J. P. (2015). Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory. *Artif. Intell.*, 226:34–74.
- [Tsarkov and Horrocks, 2006] Tsarkov, D. and Horrocks, I. (2006). Fact++ description logic reasoner: System description. *Automated reasoning*, pages 292–297.
- [Ulbricht et al., 2018] Ulbricht, M., Thimm, M., and Brewka, G. (2018). Measuring strong inconsistency. In McIlraith, S. A. and Weinberger, K. Q., editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1989–1996.
- [W3C OWL Working Group, 2009] W3C OWL Working Group (2009). OWL 2 web ontology language document overview. Technical report, W3C.
- [W3C RDF Working Group, 2014] W3C RDF Working Group (2014). Rdf 1.1 primer. Technical report, W3C.
- [W3C SPARQL Working Group, 2013] W3C SPARQL Working Group (2013). SPARQL 1.1 Query Language. Technical report, W3C.
- [Wallner, 2014] Wallner, J. P. (2014). *Complexity Results and Algorithms for Argumentation - Dung's Frameworks and Beyond*. PhD thesis, TU Vienna, Institute of Information Systems.

- [Want et al., 2015] Want, R., Schilit, B. N., and Jenson, S. (2015). Enabling the internet of things. *Computer*, 48(1):28–35.
- [Weinzierl, 2014] Weinzierl, A. (2014). *Inconsistency Management under Preferences for Multi-Context Systems and Extensions*. PhD thesis, Vienna University of Technology.
- [Weiser, 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.
- [Zaniolo, 2012] Zaniolo, C. (2012). Logical foundations of continuous query languages for data streams. In Barceló, P. and Pichler, R., editors, *2nd International Workshop on Datalog in Academia and Industry (Datalog 2.0)*, volume 7494 of *Lecture Notes in Computer Science*, pages 177–189. Springer.



---

# Work and Author Information

---

## Declaration of Authorship (Selbständigkeitserklärung)

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

---

(Ort, Datum)

---

(Unterschrift)

## **Bibliographic Data**

**Title** Multi-Context Reasoning in Continuous Data-Flow Environments

**Author** Ellmauthaler Stefan

**Year of Publication** 2018

**Number of Chapters** 8

**Number of Pages** 8 + 163

**Number of Figures** 12

**Number of Tables** 3

This thesis was typeset using the  $\text{\LaTeX} 2_{\epsilon}$  typesetting system and the graphics have been produced with PGF & TikZ.

## Academic History

### Education

since 10/2012	Doctoral study of Computer Science at the Leipzig University, Supervised by Prof. Dr. Gerhard Brewka
2009 – 2012	Master study (Dipl. Ing.) in Computational Intelligence at the Vienna University of Technology with distinction
2005 – 2009	Bachelor study (BSc) in Medicine and Computer Science at the Vienna University of Technology
2000 – 2005	Matura certificate at the HTBLuVA Wiener Neustadt in the field of informatics and business management

### Awards

07/2017	1 <sup>st</sup> Place at the 7 <sup>th</sup> Answer Set Programming Competition Modelling Competition
07/2014	3 <sup>rd</sup> Place at the Answer Set Programming Modelling Contest
08/2013	1 <sup>st</sup> Place at the 20 <sup>th</sup> PROLOG Programming Contest

### Academic Working Experience

since 10/2017	Research associate at the DFG Project BR-1817/7-2 “ <i>New tools for graph-based formal argumentation</i> ”
10/2015 – 09/2017	Research associate and administrative officer for Leipzig at the Hybris A2 Project “ <i>Advanced Solving Technology for Dynamic and Reactive Applications</i> ” for the DFG Research Unit “ <i>Hybrid Reasoning for Intelligent Systems</i> ” (FOR 1513)
10/2012 – 09/2015	Research associate and administrative officer for Leipzig at the Hybris Projects A2 “ <i>Advanced Solving Technology for Dynamic and Reactive Applications</i> ” and B2 “ <i>From Correlation to Causality: Hybrid Reasoning Over Dynamic Protein Interaction Networks</i> ” for the DFG Research Unit “ <i>Hybrid Reasoning for Intelligent Systems</i> ” (FOR 1513)

## Reviewing

Journal Reviewing	IfCoLog Journal of Logics and their Applications, Journal of Artificial Intelligence Research (JAIR)
Program Committee	AAAI 2018, COMMA 2016, SAFA 2016, IJCAI 2015 (KR Track)
Conference and Workshop Reviewing	Reviewing for IJCAI 2017, IJCAI 2016, ICLP 2016, SAFA 2016, IJCAI 2015, IULP 2015, AAAI 2014, ECAI 2014, ReactKnow 2014, ICCSW 2014, ICLP 2013, and ICCSW 2013

## Academic Activities

Guest editor of the *Theory and Practice in Logic Programming (TPLP)* special issue on *User-Oriented Logic Programming and Reasoning Paradigms*

Organiser of the *2<sup>nd</sup> International Workshop on User-Oriented Logic Paradigms (IULP 2017)*, co-located with the *14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2017)* at Espoo, Finland

Organiser of the *International Workshop on User-Oriented Logic Programming (IULP 2015)*, co-located with the *International Conference on Logic Programming (ICLP 2015)* at Cork, Ireland

Organiser of the *International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014)*, co-located with the *European Conference on Artificial Intelligence (ECAI 2014)* at Prague, Czech

## Publications and Talks

### Journals

- [1] Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer. Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. *Artificial Intelligence*, 256:68–104, 2018.
- [2] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract dialectical frameworks. an overview. *IfCoLog Journal of Logics and their Applications Volume 4, number 8. Formal Argumentation*, 4(8):2263–2317, 10 2017.



## Conferences

- [3] Stefan Ellmauthaler and Hannes Strass. DIAMOND 3.0 - A native C++ implementation of DIAMOND. In *6th International Conference on Computational Models of Argument (COMMA 2016), Potsdam, Germany, 12-16 September, 2016.*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 471–472. IOS Press, 2016.
  
- [4] Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer. Inconsistency management in reactive multi-context systems. In Loizos Michael and Antonis C. Kakas, editors, *15th European Conference on Logics in Artificial Intelligence (JELIA 2016)*, volume 10021 of *Lecture Notes in Computer Science*, pages 529–535, 2016.
  
- [5] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for computing with abstract dialectical frameworks. In Simon Parsons, Nir Oren, Chris Reed, and Federico Cerutti, editors, *5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 233–240. IOS Press, September 2014.
  
- [6] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer. Multi-context systems for reactive reasoning in dynamic environments. In Torsten Schaub, Gerhard Friedrich, and Barry O’Sullivan, editors, *21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 159–164. IOS Press, 2014.
  
- [7] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes Peter Wallner, and Stefan Woltran. Abstract dialectical frameworks revisited. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. IJCAI/AAAI, August 2013.
  
- [8] Stefan Ellmauthaler and Johannes Peter Wallner. Evaluating Abstract Dialectical Frameworks with ASP. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245, pages 505–506. IOS Press, 2012.

### Workshops

- [9] Stefan Ellmauthaler and Jörg Pührer. Stream packing for asynchronous multi-context systems using ASP. In Thomas Eiter, Wolfgang Faber, and Stefan Woltran, editors, *Proceedings of the Workshop on Trends and Applications of Answer Set Programming (TAASP 2016)*, 2016.
- [10] Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer. Towards inconsistency management in reactive multi-context systems. In Richard Booth, Giovanni Casini, Szymon Klarman, Gilles Richard, and Ivan José Varzincza, editors, *Proceedings of the International Workshop on Defeasible and Ampliative Reasoning (DARe-16) co-located with the 22th European Conference on Artificial Intelligence (ECAI 2016), The Hague, Holland, August 29, 2016.*, CEUR Workshop Proceedings. CEUR-WS.org, 2016.
- [11] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer. Mult-context systems for reactive reasoning in dynamic environments. In Stefan Ellmauthaler and Jörg Pührer, editors, *International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014)*, pages 23–30, 2014.
- [12] Stefan Ellmauthaler and Jörg Pührer. Asynchronous multi-context systems. In Stefan Ellmauthaler and Jörg Pührer, editors, *International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014)*, pages 31–38, 2014.
- [13] Stefan Ellmauthaler. Generalizing multi-context systems for reactive stream reasoning applications. In Andrew V. Jones and Nicholas Ng, editors, *Proceedings of the 2013 Imperial College Computing Student Workshop (ICCSW 2013)*, OpenAccess Series in Informatics (OASICS), pages 17–24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, September 2013.
- [14] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for argumentation: Preliminary report. In Michael Fink and Yuliya Lierler, editors, *Proceedings of the 6th International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2013)*, volume abs/1312.6140, September 2013.

### Academic treatises

- [15] Stefan Ellmauthaler. Abstract Dialectical Frameworks: Properties, Complexity, and Implementation. Master’s thesis, Technische Universität Wien, Institut für Informationssysteme, 2012.

**Other publications**

- [16] Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer. Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. *CoRR*, abs/1609.03438, 2016.
- [17] Stefan Ellmauthaler and Jörg Pührer. Asynchronous multi-context systems. In Thomas Eiter, Hannes Strass, Mirosław Trzuszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*. Springer, 2015.
- [18] Hannes Strass and Stefan Ellmauthaler. goDIAMOND 0.6.6 – ICCMA 2017 system description, 2017. Second International Competition on Computational Models of Argumentation: <http://www.dbai.tuwien.ac.at/iccma17/>.

**Additional Talks**

- [19] Asynchronous Multi-Context Systems. Stream Reasoning Workshop, Zurich, 16.01.2018.
- [20] Inconsistency Management in Reactive Multi-Context Systems. Stream Reasoning Workshop, Berlin, 8.12.2016.
- [21] Inconsistency Management in Reactive Multi-Context Systems. 8<sup>th</sup> Hybris Workshop, Dresden, 29.11.2016.
- [22] Asynchronous Multi-Context Systems 5<sup>th</sup> Hybris Workshop, Potsdam, 8.6.2015.
- [23] Generalizing Multi-Context Systems for Reactive Stream Reasoning Applications. 3<sup>rd</sup> Hybris Workshop, Dresden, 20.11.2013.
- [24] Abstract Dialectical Frameworks. 1<sup>st</sup> Hybris Workshop, Aachen, 15.11.2012.

**Poster**

- [25] Argumentation for Reactive Reasoning. Doctoral Consortium at the 14<sup>th</sup> International Conference on Principles of Knowledge Representation (KR), Vienna, 2014.
- [26] A DIAMOND in Argumentation. Advanced Course on Artificial Intelligence (ACAI), King's College, London, 2013.