Linköping Studies in Science and Technology

Licentiate Thesis No. 1783

Spatio-Temporal Stream Reasoning with Adaptive State Stream Generation

by

Daniel de Leng

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2017

This is a Swedish Licentiate's Thesis

Swedish postgraduate education leads to a doctor's degree and/or a licentiate's degree. A doctor's degree comprises 240 ECTS credits (4 year of full-time studies).
A licentiate's degree comprises 120 ECTS credits.

ISBN 978-91-7685-476-1
ISSN 0280–7971
Printed by LiU Tryck 2017

URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-138645

Abstract

A lot of today's data is generated incrementally over time by a large variety of producers. This data ranges from quantitative sensor observations produced by robot systems to complex unstructured human-generated texts on social media. With data being so abundant, making sense of these streams of data through reasoning is challenging. Reasoning over streams is particularly relevant for autonomous robotic systems that operate in a physical environment. They commonly observe this environment through incremental observations, gradually refining information about their surroundings. This makes robust management of streaming data and its refinement an important problem.

Many contemporary approaches to stream reasoning focus on the issue of querying data streams in order to generate higher-level information by relying on well-known database approaches. Other approaches apply logic-based reasoning techniques, which rarely consider the provenance of their symbolic interpretations. In this thesis, we integrate techniques for logic-based spatio-temporal stream reasoning with the adaptive generation of the state streams needed to do the reasoning over. This combination deals with both the challenge of reasoning over streaming data and the problem of robustly managing streaming data and its refinement.

The main contributions of this thesis are (1) a logic-based spatio-temporal reasoning technique that combines temporal reasoning with qualitative spatial reasoning; (2) an adaptive reconfiguration procedure for generating and maintaining a data stream required to perform spatio-temporal stream reasoning over; and (3) integration of these two techniques into a stream reasoning framework. The proposed spatio-temporal stream reasoning technique is able to reason with intertemporal spatial relations by leveraging landmarks. Adaptive state stream generation allows the framework to adapt in situations in which the set of available streaming resources changes. Management of streaming resources is formalised in the DyKnow model, which introduces a configuration life-cycle to adaptively generate state streams. The DyKnow-ROS stream reasoning framework is a concrete realisation of this model that extends the Robot Operating System (ROS). DyKnow-ROS has been deployed on the SoftBank Robotics NAO platform to demonstrate the system's capabilities in the context of a case study on run-time adaptive reconfiguration. The results show that the proposed system—by combining reasoning over and reasoning about streams—can robustly perform spatio-temporal stream reasoning, even when the availability of streaming resources changes.

This work was funded in part by the National Graduate School in Computer Science, Sweden (CUGS), the Swedish Aeronautics Research Council (NFFP6), the Swedish Foundation for Strategic Research (SSF) project CUAS, the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, and the Center for Industrial Information Technology CENIIT.

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Acknowledgements

I first and foremost want to thank my supervisor Fredrik Heintz. For someone who planned half a decade ahead, remaining in Sweden was not something I had considered. Your research on DyKnow and the sense-reasoning gap brought me to a beautiful Swedish winter in November of 2012. I was planning to do my Master's thesis work in Linköping before attending a graduate school in the United States to be with the person I love. You gave me an opportunity which brought him to Sweden instead and allowed me to continue working on interesting problems here at AIICS. It changed our lives for the better, and for that I am forever grateful to you. As a student, I also want to thank you for your time and support, and I look forward to our future challenges and opportunities.

Thanks to Patrick Doherty for taking the time to read through an earlier version of this thesis. It can be difficult to see problems with the writing when one is in the middle of it. Your comments were very valuable in improving the quality of the thesis.

I also want to thank Mattias Tiger for our daily (sometimes frustrating but healthy) discussions about everything research and outside of research, and above all, for being a friend and a guide to all things Sweden. Our lunches and fikas together with Jon Dybeck and Erik Hansson are always a great way to disconnect from work or give a fresh perspective.

To everyone at AIICS; I am grateful for our Friday fikas, your continuing feedback and support in matters research and teaching, and your company. Thank you all.

Last but not least, I want to thank my beloved husband Riley, my parents Eric and Natasha, my siblings Samantha and Daryl, and my parents-in-law Paige and Gary for their love and support across borders and oceans. You make an enormous difference.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-world robotic systems must be able to interpret and reason about uncertain sensor observations to effectively operate in the physical world. Such observations occur in the context of and across time and space. Consequently, observations are temporally and spatially connected to each other. The discrete observations succeed each other like snapshots that, when taken together, tell us a story about the world we reside in. *Stream reasoning* is a subfield of Artificial Intelligence (AI) that focuses on the incremental reasoning over incrementally-available information, which we characterise as *streams* containing state information. More specifically, stream reasoning is a subfield of Knowledge Representation (KR), which is itself a subfield of AI. For many systems, including autonomous robotic systems, this streaming information is generated from sensor observations. Stream reasoning plays an increasingly important role as robots are no longer confined to carefully crafted environments and instead have to deal with the highly-dynamic physical world that is inhabited by other entities. In such a context, *situation awareness* is of great importance and covers not just the environment outside of an autonomous robot, but also how its percepts arise from its own controllable internal configuration.

The contributions presented in this thesis can be roughly divided into two distinct but adjoining strands; *spatio-temporal stream reasoning* and *adaptive state stream generation*. Spatio-temporal stream reasoning is an application of reasoning *over* streams, which means reasoning with the streaming data that makes up a stream. Streaming data is assumed to become incrementally available over time, which makes it fundamentally different from database contexts. Examples of (incremental) reasoning over streams include the generation of a stream of more refined data, or the drawing of conclusions from streaming data. Adaptive state stream generation utilises reasoning *about* streams, which can be regarded as meta-stream reasoning. In this view, the streams themselves—and by extension, their properties—are of interest for the purpose of reasoning. Both views are complemen-

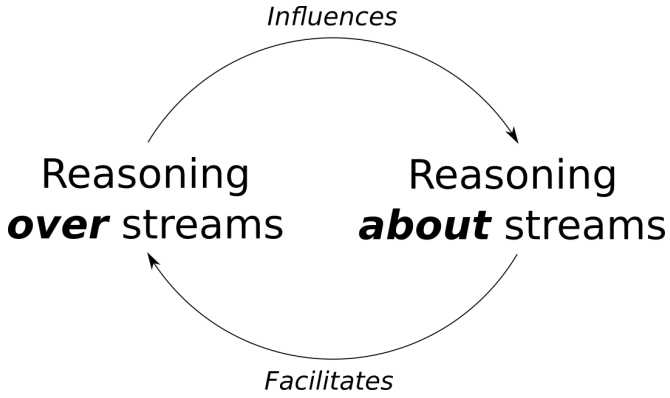

Figure 1.1: Synergy effect between reasoning over streams and reasoning about streams.

tary and form the basis for the two strands of this thesis. As illustrated in Figure 1.1, reasoning *about* streams can facilitate and strengthen reasoning *over* streams, and reasoning *over* streams can influence the reasoning *about* streams. The two strands thus provide a natural synergy effect wherein the whole is greater than its individual parts.

Spatio-temporal stream reasoning. Many approaches towards temporal stream reasoning exist, but there have been fewer attempts at integrating spatial and temporal streaming information. The incorporation of spatial information can be of great importance when reasoning about the physical world, such as is the case for autonomous robots. This thesis focuses on logic-based spatio-temporal stream reasoning, where logic formulas are evaluated over streams using a technique called *progression*, further explained in Chapter 3. Logic formulas allow us to very clearly specify crisp statements that describe the temporal and spatial properties of the world. *Execution monitoring* is an application of progression in which we check whether the expected properties hold or are violated through formula evaluation. This is an important ability for safety in autonomous robotics. Contributions in the area of spatio-temporal stream reasoning (i.e. reasoning over streams) make up the first part of this thesis.

Adaptive state stream generation. Recently there has been a lot of progress in the development of stream reasoning systems. The number of sources for streams—such as sensors or Internet of Things (IoT) devices—is increasing, yet most research assumes that the streaming resources are fixed and known. Therefore, while it is important to reason about which streaming resources to subscribe to, most of today's systems lack the capability to do so. We argue that it is unreasonable to assume that the streaming resources

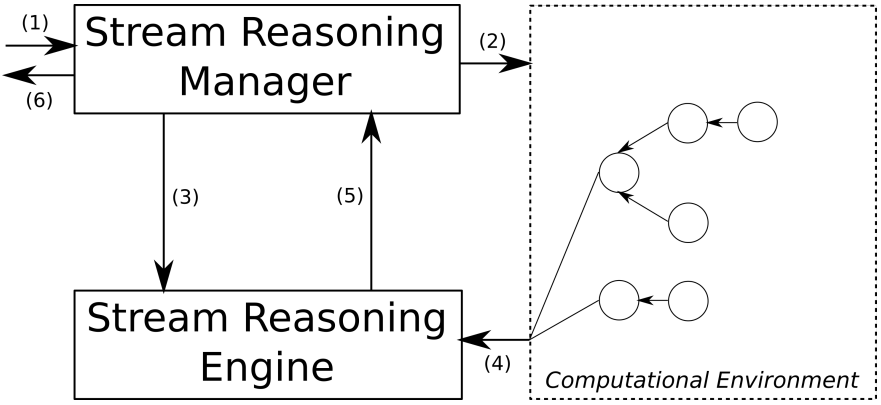

Figure 1.2: Conceptual system overview for spatio-temporal stream reasoning with adaptive state stream generation.

are fixed and known, and that being able to reason about these dynamics is important for autonomous systems in order to effectively operate in the real world. This thesis focuses in particular on the problem of reliably generating a stream containing states (i.e. a *state stream*) for the evaluation of logic formulas, where the computational resources may change over time. This is done by reasoning about streams, and in particular how streams can be generated.

Figure 1.2 shows a conceptual system overview combining the two strands, which we refer to in further detail in the corresponding chapters. The strand on spatio-temporal stream reasoning focuses on the evaluation of logic formulas; the strand on adaptive state stream generation subsequently focuses on adaptive reconfiguration of computational environment in order to maintain a stream used for the evaluation of such a formula. The numbers in the conceptual system overview identify interactions. First, a formula is provided to the stream reasoning manager (1), together with specifications of stream-generating components and their semantic descriptions. This formula acts as a query; the task of the system is to determine the truth value of the formula. To evaluate the formula, the manager needs to configure stream-generating components (2) in such a way that they produce a stream of states that can be used for this purpose. Then the formula can be added to the stream reasoning engine (3), which can subscribe to the state stream produced by the computational environment (4). With a formula and a suitable state stream, the engine can perform progression until it determines the truth value of the formula, which is subsequently sent back to the manager (5). The manager can internally act on the evaluation result if applicable, and subsequently sends the result back to the client (6). For a more concrete example, consider the following synergy examples that

illustrate the framework's behaviour.

Example 1 (Adapting to changing conditions)**.** *Suppose that we have a robot that is tasked with the visual tracking of objects in an environment. Its preferred tracker assumes well-lit surroundings to produce high-precision tracks. It also has a redundant tracker that provides lower-quality tracks but it does not require the environment to be as well-lit. An execution monitor can send a formula of the form 'If I am in my room, it is always the case that if the light condition of my surroundings is poor, then the surroundings will be well-lit within 10 seconds.' to the stream reasoning manager. If the formula evaluates to false, for example because it has become dark because the room's lights switched off and nobody is there to turn them back on, this triggers the monitor to swap out the two trackers, which will result in a reconfiguration. Any formulas depending on a stream of tracks are progressed with minimal interruption.*

The situation described above uses the progressor to check the changing state of the environment. The formula allows us to very precisely specify the conditions in which the preferred tracker should be used; in the robot's room and in well-lit conditions. The 10 second window ensures that short interruptions of the lighting conditions do not lead to reconfigurations. The formula is provided to the stream reasoning manager, which is assumed to already have access to specifications for stream-generating components. It uses the information in the grounding to reconfigure the system such that the light conditions and robot position are provided in a stream. This stream is fed to the stream reasoning engine, which uses it to evaluate the provided logic formula. If the formula evaluates to false, we know that the tracker's assumed well-lit environment no longer holds, so the stream reasoning engine tells the stream reasoning manager that it can no longer be used. The stream reasoning manager subsequently reconfigures the system by using the alternative tracker instead.

Example 2 (Introspection)**.** *Suppose that we have a robot system consisting of a number of components that refine streaming information. Every component in addition produces a stream with computational resource usage in terms of CPU time and memory consumed. While our components were designed to be load-efficient, we want to ensure that changes made during the development do not result in unexpected behaviour. The stream reasoning manager is therefore sent a formula stating 'The resource usage of all components is nominal.' After one developer commits their changes, a software test is run. While all feature tests pass, the formula evaluates to false; the changes violate the resource constraints. The execution monitor upon noticing this violation swaps out the offending module for an older version.*

In the above example, the operational environment used for sensing is a robot's internal processing, which is reasoned over by the stream reasoning engine. The formula envaluation is being used to detect unwanted situations and to respond to those situations through reconfiguration.

1.1 Scope and limitations

The aim of this Licentiate thesis is

> **to combine spatial and temporal stream reasoning techniques based on logic, and to develop techniques for the adaptive generation of streams needed for this type of reasoning.**

The two strands are considered separately (as before) for the purposes of identifying scope and limitations, before being joined into a single integrated system. The aim as described covers all components previously presented as part of the conceptual overview, with the exception of the stream reasoning manager acting on evaluation results by performing reconfigurations. While doing so is made possible as the result of the work presented in this thesis, an in-depth exploration is left for future work.

Spatio-temporal stream reasoning. There are many ways to combine spatial and temporal stream reasoning. This thesis focuses specifically on logic-based stream reasoning techniques that combine these two aspects. For the spatial reasoning we focus on topological spatial modeling and inference. Combining spatial and temporal logics is not something new—Kontchakov et al. (2007) provide a lengthy overview of different techniques that have been employed to combine the two. However, the application of these techniques to the domain of stream reasoning requires a different approach, which is what is presented in this thesis. The goal of this thesis strand is thus to design a spatio-temporal logic that is applicable to stream reasoning, which requires a form of incremental path checking.

Adaptive state stream generation. State stream generation deals with the incremental generation of logical interpretations (states) for each time point. There are many ways this could be done. However, it is usually ignored when the topic of interest is the temporal logic, because it is generally assumed that the states exist. The goal of this thesis strand is to design a method for the adaptive, on-demand generation of interpretations grounded in an underlying stream processing environment. This is closely related to research areas such as service composition or configuration planning, albeit with a focus on repair. The thesis does not seek to introduce novel languages for describing the relationship between entities in a stream processing environment; areas such as the semantic web have worked on related problems. Instead, the thesis focuses on the problem of adaptively reconfiguring the stream processing environment to deal with unexpected changes. Subscriptions to such streams are called semantic subscriptions in this context.

Integration. The resulting techniques are integrated into a stream reasoning framework for the purpose of supporting autonomous robots in their

operations. The resulting system should be able to leverage pre-existing modules, or it should be possible to convert those modules with minimal effort. The focus is thus on the usability of the resulting system towards research into autonomous robots. This means that a custom, specialised implementation is unlikely to be sufficiently suitable. Further, the scope of this thesis limits itself to the unidirectional support from adaptive state stream generation to spatio-temporal stream reasoning. The described (bidirectional) synergy effect by allowing the stream reasoning to affect the adaptive state stream generation is the focus of ongoing research.

1.2 Methodology

The procedure used to produce the aforementioned contributions is based on the need to offer real-world applicable solutions to real-world problems. To this effect, the work towards this thesis can be categorised into three categories; *theory*, *engineering*, and *deployment*.

Theory. First, theoretical contributions were developed and proposed, providing a solid foundation that doubles as a clear design specification. These theoretical contributions are based on and extend previous work in the various fields. The spatio-temporal stream reasoning strand is closely related to research in the field of knowledge representation and reasoning, for example.

Engineering. The different theoretical results were tested empirically as software artefacts. While the contributions themselves are general and could be implemented in a variety of ways, the goal of this work was to provide a stream reasoning framework implementation that integrates these results in a useful manner. This presented a number of engineering problems that were resolved as part of the integration work. The engineering work focused in part on the applicability of the resulting software artefacts. Special care was taken to make sure that the software was easy to use by other developers, decreasing the cost of adoption. The engineering efforts often highlighted potential theoretical problems which had to be resolved.

Deployment. The resulting software artefacts were deployed on the SoftBank Robotics[1] NAO robot platform. Since the work on state stream generation relies on underlying implemented functionality, software under development for the RoboCup Standard Platform League (SPL) was used and adapted to work with the stream reasoning framework. This presented an interesting test-bed for testing the ease of integration, and highlighted various engineering problems that required solving. The result of deployment

[1] Formerly called Aldebaran; acquired by SoftBank Robotics in 2015 and renamed accordingly on May 19th, 2016.

often also yields or highlights interesting theoretical questions and problems.

The theoretical foundation thus provide a basis upon which the proposed system is built. While some of the presented results are purely theoretical in nature, the focus lies on robotics-related application domains. By providing a formal model of the system, the results can therefore be reproduced in other system realisations than the one presented in this thesis, using different platforms than those used here. This supports the claim of generality of these results. Lastly, the goal is to make our system realisation open-source in the future, allowing for our empirical evaluations to be repeated.

1.3 Contributions

The contributions of this thesis are:

1. A spatio-temporal logic MSTL was developed based on Metric Temporal Logic (MTL) and the Region Connection Calculus (RCC-8). Subsequent improvements to progression were formalised and empirically evaluated. Further, the application of the 'next' operator to spatial terms, as originally introduced as part of the logic ST_1, was extended with efficient rewriting rules so it can be used for stream reasoning. These contributions are based on material that originally appeared in Heintz and de Leng (2014) and de Leng and Heintz (2016a).

2. A formal model of a distributed stream reasoning framework was developed, along with the formalisation of its dynamics in terms of changes to the computational environment. Reconfiguration of the computational environment allows for the generation of streams based on requests, for example to support the evaluation of a logic formula. An adaptive reconfiguration algorithm is presented. To support adaptive reconfiguration planning, the cost of using the framework's components is learned on-line. These contributions are based on material that originally appeared in Heintz and de Leng (2013) and de Leng and Heintz (2014, 2015a,b, 2017).

3. The DyKnow-ROS[2] dynamically reconfigurable stream reasoning framework was implemented as an extension to the Robot Operating System (ROS). The required reconfigurability strengthens ROS, which by default does not support this ability. ROS visualisation tools were enhanced with the ability to visualise the dynamically-changing environment. These contributions are based on material originally appeared in de Leng and Heintz (2016b).

[2]DyKnow homepage: `http://www.dyknow.eu`

The complete listing of publications covered in this thesis is as follows:

- **D. de Leng** and F. Heintz. Towards Adaptive Semantic Subscriptions for Stream Reasoning in the Robot Operating System. To appear in *Proceedings of the 30th IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.
- **D. de Leng** and F. Heintz. DyKnow: A Dynamically Reconfigurable Stream Reasoning Framework as an Extension to the Robot Operating System. In *Proceedings of the 5th IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2016.
- **D. de Leng** and F. Heintz. Qualitative Spatio-Temporal Stream Reasoning With Unobservable Intertemporal Spatial Relations Using Landmarks. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016.
- **D. de Leng** and F. Heintz. Ontology-Based Introspection in Support of Stream Reasoning. In *Proceedings of the 13th Scandinavian conference on Artificial Intelligence*, 2015.
- **D. de Leng** and F. Heintz. Ontology-Based Introspection in Support of Stream Reasoning. In *Proceedings of the 1st Joint Ontology Workshops held at the 24th International Joint Conference on Artificial Intelligence*, 2015.
- F. Heintz and **D. de Leng**. Spatio-Temporal Stream Reasoning with Incomplete Spatial Information. In *Proceedings of the 21st European Conference on Artificial Intelligence*, 2014.
- **D. de Leng** and F. Heintz. Towards On-Demand Semantic Event Processing for Stream Reasoning. In *Proceedings of the 17th International Conference on Information Fusion*, 2013.
- F. Heintz and **D. de Leng**. Semantic Information Integration with Transformations for Stream Reasoning. In *Proceedings of the 16th International Conference on Information Fusion*, 2013.

Lastly, one contribution was the invited publication of a popular science article (not peer-reviewed) titled "Querying flying robots and other Things: Ontology-supported stream reasoning" in XRDS (de Leng, 2015) within the theme of the Internet of Things (IoT) targeted at students within the field of computer science.

1.4 Outline

The remainder of this thesis is organised as follows. Chapter 2 covers stream reasoning preliminaries. In Chapter 3, we introduce the spatio-temporal logic MSTL which combines temporal and spatial reasoning. The

logic is complemented with efficient progression techniques to make it applicable to stream reasoning, and an empirical evaluation is provided to support this claim. Chapter 4 discusses a formal model for a stream reasoning framework that can adaptively generate state streams for formula evaluation. An implementation combining these contributions is presented in Chapter 5, where the DyKnow stream reasoning framework is introduced and empirically evaluated. Chapter 6 describes a case study in which DyKnow is used on NAO robots. A survey of related work is presented in Chapter 7 and compared to the thesis contributions. Finally, Chapter 8 concludes the thesis and discusses future work.

Chapter 2

Stream reasoning

Stream reasoning spans a broad research area and is the focus of this thesis. In this chapter, the concepts of streams and stream reasoning are briefly explained. In particular, the thesis focuses on execution monitoring as an application of stream reasoning. This chapter introduces path checking and relates it to progression, which is the technique of our choice for formula evaluation. Progression is used as a tool to perform execution monitoring. This chapter subsequently gives an overview of stream reasoning systems, considering different families, before giving a brief history of the DyKnow project and its associated stream reasoning systems.

2.1 Introduction

Stream reasoning does not have a singular definition in the literature, but variations generally agree with the notion that stream reasoning is some kind of reasoning pertaining to incrementally-available information. In this thesis, the concept of *stream reasoning* is therefore defined as follows.

Definition 1 (Stream reasoning)**.** Stream reasoning *is the incremental reasoning* over *and* about *streams of incrementally-available information.*

The incremental nature of streams constitutes a significant departure from classical databases. One analogy is that of a waterfall flowing into a basin: Classical approaches operate on the basin, whereas stream reasoning approaches operate on the waterfall. Classical approaches thus only operate on what is stored and always on everything that is stored, whereas stream reasoning puts constraints on how much can be stored and always assumes to only have a fragment of the entire stream to operate on. Due to the properties of streaming data, new approaches are needed. Laney (2001) originally described the terms volume, velocity and variety as important properties for describing data. These properties were subsequently

extended to define Big Data. The following stream properties originate from the 'four Vs of big data' applied to a stream reasoning context:

Volume. One can no longer assume that the data can be collected in its entirety prior to processing it. The volume of data may simply be too large for any practical storage to take place. Streaming data is therefore generally assumed to be accessed once and then lost.

Velocity. The incremental nature of streams invokes the property of velocity, i.e. how quickly data becomes available. Depending on the source of a data stream one can or cannot make assumptions about its velocity. For example, user-generated content can be highly irregular and bound to human behavioural patterns, whereas sensor data in a real-time system can be assumed to have a fixed frequency. A general stream reasoning system must be able to cope with differences in velocity, and high velocity in particular.

Variety. Streaming data can originate from many heterogeneous sources in various data formats and as various data types. Examples of different data types are text, images, and speech. Being able to interpret the data from streams in various formats and types is important in order to effectively work with this data.

Veracity. The trustworthiness and accuracy of data is another important factor to consider when dealing with streaming data. The trustworthiness of data is in part based on who produced the data and who provided it; some sources may be of poor quality or (purposely or not) misrepresent information. This may also be a consequence of low accuracy of data.

Different stream reasoning systems focus on different aspects. For social media tools, variety and veracity may be far less important than dealing with volume and velocity, as the focus is user-generated unstructured data. In robot systems, value, veracity and velocity are important in order to deal with a rapidly-changing environment. This leads to different perspectives on stream reasoning.

2.1.1 Temporal logics

Logic-based approaches to stream reasoning are often related to temporal (modal) logics such as Linear Time Logic LTL, Computation Tree Logic CTL, or their combination CTL*. The syntax and semantics for propositional LTL are as follows.

Definition 2 (LTL syntax)**.** *The syntax for propositional LTL is as follows for atomic propositions $p \in \mathsf{Prop}$ and well-formed formulas (wffs) ϕ and ψ:*

$$p \mid \neg\phi \mid \phi \vee \psi \mid \phi \,\mathcal{U}\, \psi \tag{2.1}$$

The semantic of LTL are based on *temporal models* which describe a sequence of time-points. These temporal models are represented as Kripke models.

Definition 3 (Temporal model)**.** *A* temporal model *is a Kripke model $\mathcal{M} = \langle T, <, V\rangle$ where T denotes a set of time-points, $<$ denotes a temporal ordering over T, and $V : T \mapsto 2^{\mathsf{Prop}}$ denotes a mapping from time-points to states.*

The temporal model thus represents a time-line. The semantics of LTL are as follows.

Definition 4 (LTL semantics)**.** *The LTL statement that a wff ϕ holds in a temporal model $\mathcal{M} = \langle T, <, V\rangle$ at time-point $t \in T$ is defined recursively:*

$$\mathcal{M}, t \models p \text{ iff } p \in V(t) \text{ for } p \in \mathsf{Prop} \tag{2.2}$$
$$\mathcal{M}, t \models \neg\phi \text{ iff } \mathcal{M}, t \not\models \phi \tag{2.3}$$
$$\mathcal{M}, t \models \phi \vee \psi \text{ iff } \mathcal{M}, t \models \phi \text{ or } \mathcal{M}, t \models \psi \tag{2.4}$$
$$\mathcal{M}, t \models \phi \,\mathcal{U}\, \psi \text{ iff } \exists t \leq t' : \mathcal{M}, t' \models \psi \tag{2.5}$$
$$\text{and } \forall t'' : t \leq t'' < t' : \mathcal{M}, t'' \models \phi$$

The syntax of propositional LTL is often extended to include the following Boolean operators:

$$\top \equiv_{def} p \vee \neg p \tag{2.6}$$
$$\bot \equiv_{def} \neg\top \tag{2.7}$$
$$\phi \wedge \phi \equiv_{def} \neg(\neg\phi \vee \neg\psi) \tag{2.8}$$
$$\phi \rightarrow \psi \equiv_{def} \neg\phi \vee \psi \tag{2.9}$$
$$\phi \leftrightarrow \psi \equiv_{def} (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \tag{2.10}$$
$$\phi \,\mathcal{R}\, \psi \equiv_{def} \neg(\neg\phi \,\mathcal{U}\, \neg\psi) \tag{2.11}$$
$$\Diamond\phi \equiv_{def} \top \,\mathcal{U}\, \phi \tag{2.12}$$
$$\Box\phi \equiv_{def} \neg\Diamond\neg\phi \tag{2.13}$$

We refer to these extensions as *syntactic sugar* since they do not occur formally in the above semantics. The symbols $\Box$, $\Diamond$, $\mathcal{U}$ and $\mathcal{R}$ are called 'always', 'eventually', 'until', and 'release' respectively. Using LTL with the syntactic sugar provided by the above operators, we can formally describe temporal statements in the logic.

Example 3 (LTL statement)**.** *Consider the statement from a previous synergy example: "If I am in my room, it is always the case that if the light condition of my*

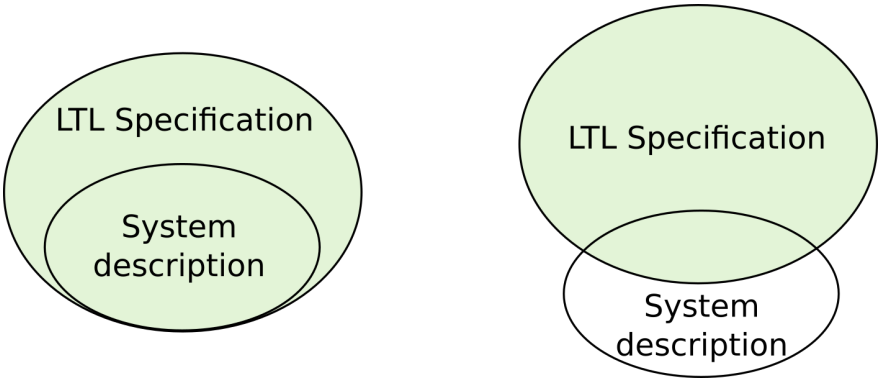

Figure 2.1: Left: All models of the system description are also models of the LTL specification, showing correctness. Right: Some models of the system description are not models of the LTL specification, indicating that the specification is violated by some system traces.

surroundings is poor, then the surroundings will be well-lit within 10 seconds." This statement can be approximated as

$$\mathsf{inRoom} \rightarrow (\Box(\mathsf{poorLight} \rightarrow \Diamond \mathsf{goodLight})), \tag{2.14}$$

where propositions inRoom, poorLight *and* goodLight *stand for "I am in my room", "the light condition of my surroundings is poor", and "the surroundings are well-lit" respectively. Note that this is an approximation since LTL is not able to describe metric conditions such as "within the next 10 seconds."*

Many extensions of LTL exist. CTL is sometimes also referred to as *branching-time logic* due to its branching time-lines. It allows for reasoning over paths in a tree structure and separates state formulas from path formulas. CTL* is a combination of CTL and LTL, and has the expressivity of both combined. Another extension of LTL is Metric Temporal Logic (MTL), which allows for metric constraints on the modal operators, restricting them to a specific time-period. This allows for statements such as "ϕ is true for the next 10 time-points," or "ψ becomes true within the next 10 time-points."

2.1.2 Automata-theoretic model checking

LTL has proven to be extremely useful in areas such as formal verification, in which the correctness of a program is tested by comparing a description of the program to an LTL specification of how the program should behave. If the description falls within the scope of the specification, the program is correct. This relation is illustrated in Figure 2.1. On the left, all models of the system description are also models of the LTL specification,

which means that the system verifiably operates within the constraints described by the specification. On the right, there are also models of the system description that occur outside the LTL specification, meaning they violate the specification. Automata-theoretic model checking makes use of ω-automata to describe a program in terms of possible state sequences, which can be regarded as streams. These finite automata operating on (infinite-length) *ω-words* are therefore sometimes called 'stream automata'. Finite automata operate on finite-length words $w \in \Sigma^*$, where Σ^* denotes the set of finite strings over a finite alphabet Σ. For infinite-length words we instead use the set Σ^ω of ω-words $\alpha \in \Sigma^*$, from which languages of infinite words can be constructed. Just as regular languages can be described by regular expressions, ω-regular languages can be described by ω-regular expressions.

Example 4 (Finite and infinite regular languages)**.** *Suppose we have a finite alphabet $\Sigma = \{a, b\}$. The regular expression $a(a|b)^*$ describes any finite sequence of a's or b's following a single a. These sequences describe finite-length words. We can describe ω-words with ω-regular expressions. As an example, consider the ω-regular expression a^*b^ω, which describes all ω-words which start with a finite sequence of a's followed by an infinite sequence of b's.*

Finite automata use accept (or final) states to determine the acceptability of words: if a word ends in an accept state of a given finite automaton, the word is considered to be *accepted* by that finite automaton. However, since ω-words do not have a final element, the acceptance conditions of ω-automata differs from those of finite automata. Different types of ω-automata consequently exist with varying semantics in terms of acceptance conditions, but *Büchi ω-automata* are commonly used.

Definition 5 (Büchi automata)**.** *A* Büchi automaton *$\mathcal{B}$ is a type of ω-automaton over an alphabet Σ denoted by a tuple*

$$(\Sigma, Q, \Delta, Q^0, F), \tag{2.15}$$

where Q denotes a finite set of states, $Q^0 \subseteq Q$ denotes the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ denotes the transition relations, and $F \subseteq Q$ denotes the set of accepting states. An ω-word $\alpha \in \Sigma^\omega$ is accepted by $\mathcal{B}$ iff for its associated run σ it is the case that $\mathsf{inf}(\sigma) \cap F \neq \varnothing$, i.e. at least one of the accept states is encountered infinitely often.

ω-automata encode a set of ω-words. The concept of a *run* formally describes such ω-words.

Definition 6 (Run)**.** *A* run *on an automaton is an ω-word σ for language Q, starting with $\sigma(0) \in Q^0$ and $(\sigma(i), \alpha(i), \sigma(i+1)) \in \Delta$ for every $i \geq 0$, i.e. runs start in an initial state and only contain valid transitions between states. The function $\mathsf{inf} : \sigma \mapsto Q$ is used to denote the set of states that occur infinitely often.*

We can describe a system in terms of a Büchi automaton $\mathcal{B}_{sys}$ such that the set of ω-words that are accepted by $\mathcal{B}_{sys}$ correspond to the set of possible system traces. A Büchi automaton thus describes a language $\mathcal{L}(\mathcal{B}) \subseteq \Sigma^{\omega}$. LTL is commonly used to describe the properties of a system which we want to verify. These properties can then be translated into equivalent Büchi automata in various ways, by converting the Kripke model for an LTL specification into a Büchi automaton such that $\Sigma = 2^{|\mathsf{Prop}|}$. A survey of techniques is presented by Vardi (2007). The conversion of a specification ϕ results in a Büchi automaton $\mathcal{B}_{\phi}$. If we can determine that $\mathcal{L}(\mathcal{B}_{sys}) \subseteq \mathcal{L}(\mathcal{B}_{\phi})$, we prove that the system adheres to the formal LTL specifications.

The approach used for formal verification implicitly models the set of all valid interpretations for an LTL statement by encoding them into an ω-automaton. This technique can also be used for logic-based stream reasoning where we want to determine the truth value of an LTL formula ϕ given a stream. Such a stream corresponds to an ω-word α. By encoding the LTL formula into a Büchi automaton $\mathcal{B}_{\phi}$, the task is to determine whether $\alpha \in \mathcal{L}(\mathcal{B}_{\phi})$. This is however a costly task, as the resulting automaton is in the worst-case exponential in the size of the formula, although many optimisations exist. Checking whether a trace is accepted by such a constructed automaton (i.e. whether $\alpha \in \mathcal{L}(\mathcal{B}_{\phi})$) is then linear in the size of the formula in the worst-case.

2.1.3 Progression-based path checking

For applications such as run-time verification, we are not interested in finding *all* models that satisfy a formula. Rather, given an incremental sequence of observations, the challenge is to determine whether the observations satisfy the formula. This is referred to as *path checking*. Path checking can be useful to guard the behaviour of an autonomous system and can be used to detect e.g. software bugs or nefarious code.

Path checking can be performed using automata-theoretic model-checking techniques such as the one shown earlier. A different approach to path checking is based on an incremental checking technique called *progression* (Bacchus and Kabanza, 1996, 1998). A formal presentation of progression is deferred to Chapter 3. Informally, progression can be used to incrementally check whether the states received thus far make the checked-against formula true or false. Progression achieves this by rewriting the formula. When receiving a state, the formula is decomposed into a part for the current time-point and a remainder. By applying the current state to its constituent subformula, the entire formula can be made true or false regardless of any future states, and we may terminate early. Path checking through progression is useful in for example execution monitoring, where we monitor the behaviour of a system relative to rules that are specified in a logic. If the evaluation of these rules returns 'false', we conclude that the system

has violated a predefined rule and must take (immediate) action to recover to a legal state.

One major advantage of using progression instead of automata is that it bypasses the preprocessing cost associated with building an automaton from a formula. The syntactic rewritings performed by progression have a time complexity linear in the size of the formula, but the formula can grow exponentially large in the worst case. In many practical cases, however, the formula growth appears to be managable and we avoid an exponential blow-up associated with generating an automaton. Therefore, progression is used for path checking in the context of this thesis.

2.2 Stream reasoning systems

Stream reasoning systems are systems that are designed to perform reasoning with streams. Cugola and Margara (2012) collectively refer to stream reasoning systems as *Information Flow Processing* (IFP) systems, and provide an excellent survey of the various approaches. The following is a short contrast between two classes of stream reasoning systems; the *Data Stream Management* (DSM) systems[3], and the *Complex Event Processing* (CEP) systems. The boundaries between DSM and CEP systems can however be blurry at times.

2.2.1 DSM systems

Data Stream Management Systems (DSMS) originate from the area of databases and Database Management Systems (DBMS). DSMS take continuous queries that produce results for the duration that they are active. An early example of this is the Stanford Stream Data Manager (STREAM) (Arasu et al., 2004) which supported the Continuous Query Language (CQL) (Arasu et al., 2003, 2006). CQL is based on the Structured Query Language (SQL) and extends it with windowing operations. The use of windowing operations allow DSMS to convert *streams to relations* (S2R) and *relations to streams* (R2S). This means that an input stream is never considered in its entirety; instead, the most recent few values are considered in accordance with a window rule (e.g. sliding or tumbling windows) and a relation is repeatedly constructed from the input stream. DSMS querying languages then allow for traditional DBMS operations to be applied to these relational structures, and the resulting relations are converted back into an output stream.

More recently, stream reasoning has become and area of interest within the Semantic Web (SW), which seeks to make the web machine-readable (Berners-Lee et al., 2001). Data in the Semantic Web is often represented in the Resource Description Format (RDF). A popular querying language for

[3]The term 'Data Stream Management Systems' is commonly written as DSMS, but when contrasted with 'CEP systems' it is also written as 'DSM systems'.

RDF data is the SPARQL Protocol and RDF Query Language (SPARQL). Similar to CQL providing a continuous extension of SQL, Barbieri et al. (2009) present Continuous SPARQL (C-SPARQL) as a continuous extension of SPARQL by employing window operations for sliding and tumbling windows. Semantic Web approaches commonly relate RDF data to an ontology, and hence add operations for *relations to ontologies* (R2O) and *ontologies to relations* (O2R) in addition to the operations for converting between streams and relations.

The use of windows for stream reasoning has been studied by Beck et al. (2014, 2015) as part of a logic-based formalisation of stream reasoning. One result is the logical window operator $\boxplus$, which describes the semantics of a large variety of windowing operations that can be applied to streams.

2.2.2 CEP systems

The focus for CEP systems is on *events* and combinations of events. An event is sometimes broadly defined as 'anything that happens or is perceived as happening.' Events can be used to refer to single time-points, but also to time intervals. One key difference between CEP systems and DSM systems is how values are temporally related. Whereas DSM systems make use of windows, CEP systems tend to make use of temporal orderings. The detection of a queried temporal ordering of events can itself be seen as a complex event.

Early CEP techniques include *chronicle recognition* systems, which were introduced by Ghallab (1996). Chronicles are represented by (complex) events and metric temporal constraints on those events. Chronicles can be detected in a stream by checking for the occurrence of their composite events relative to the metric temporal constraints.

Gyllstrom et al. (2006) present the SASE language for complex event pattern matching, which is further extended by Diao et al. (2007) into SASE+. SASE and SASE+ make use of windows to restrict pattern lengths, but rather than aggregating data with relational operations from DBMS, they check for the presence or absence of events in sequences of interest. A commonly used example for SASE is that of RFID tags for store products to keep track of their location events; some sequences of events may indicate theft or item misplacement.

The Semantic Web also supports CEP querying languages. One example is yet another continuous query extension of SPARQL proposed by Anicic et al. (2011). Event Processing SPARQL allows for CEP queries similar to SASE and SASE+ by checking for the (optional) sequential occurrence of events.

CEP systems thus do not exclude windows, but focus primarily on the temporal relations that exist between events. When a complex event pattern is matched, this may be used as a complex event detection yielding a resulting event stream.

2.3 Project DyKnow

The DyKnow project has focused on stream reasoning for several years, leading to several generations of DyKnow artefacts, some of which were successfully deployed in larger autonomous UAV systems. Initially DyKnow was an acronym for *Dynamic Knowledge Processing*. This was later extended to *Dynamic Knowledge Processing and Object Management*. Presently, DyKnow is a pseudo-acronym.

We distinguish between three generations of DyKnow, where the latest generation of DyKnow is the product of the research leading up to this thesis. The first generation of DyKnow was first published as part of Heintz and Doherty (2004) and finalised as part of Heintz (2009). Its implementation was done using the Common Object Request Broker Architecture (CORBA), and primarily focused on the manipulation and abstraction of streaming data in the early years of stream reasoning and stream processing research. In this thesis we will refer to this version of DyKnow as *DyKnow-CORBA* if there is a risk of ambiguity.

The second generation of DyKnow came about in part due to the switch to ROS. The initial conversion was performed as part of the Master's thesis work by Dragisic (2011); Lazarovski (2012); Hongslo (2012) and a subsequent publication by Heintz and Dragisic (2012). In these works, DyKnow's stream processing capabilities were converted to ROS. These capabilities were then extended with initial support for RCC-8 and an ontology-based method for connecting logical symbols to streams. Finally, de Leng (2013) sought to unify the above disparate components and further extended the semantic matching capabilities of DyKnow resulting in the work presented in Heintz and de Leng (2013). The unification process exposed some critical shortcomings of DyKnow: the semantics of the stream processing languages were difficult to express; the stream processing focused on streams and initially ignored transformations; and the state stream generation yielded extremely long queries for simple operations. The decision was made to instead focus on transformations, from which streams are a product.

The third generation of DyKnow thus focuses on the management of streams by managing their producing transformations. Since ROS was used, it needs to be possible for ROS-based implementations to be integrated into DyKnow with minimal or no effort. Part of these efforts resulted in an extension to ROS to cope with some of its shortcomings. This latest version of DyKnow is called *DyKnow-ROS*.

We provide a detailed overview of the third-generation DyKnow stream reasoning framework (i.e. DyKnow-ROS) in Chapters 3–5.

2.4 Summary

In this chapter, we defined streams to be incrementally-available sequences of time-stamped values, and stream reasoning as reasoning over those sequences. One can roughly subdivide stream reasoning systems into two categories. DSM systems often focus on window-based aggregation of streaming data; CEP systems focus primarily on atomic events and combinations of events. Streaming data can be characterised using a set of properties based on the '4 Vs of big data'; *volume*, *velocity*, *variety*, and *veracity*. For the purposes of robotics applications, value, veracity and velocity are of particular importance in order to deal with rapidly-changing environments observed by sensors. To assist in situation awareness in order to deal with such an environment, we are especially interested in execution monitoring applications. One common approach is to present rules of interest in terms of LTL and to generate corresponding automata for model checking. An alternative approach is to apply syntactic rewriting rules in a procedure called progression, which is the approach pursued in this thesis.

Chapter 3

Spatio-temporal stream reasoning

Logic-based stream reasoning commonly makes use of temporal logics to express statements concerning the truth value of properties over time. Similar to temporal statements, many autonomous robotic systems can also benefit from or require the ability to make statements concerning spatial properties. This chapter includes and extends previously published material (Heintz and de Leng, 2014; de Leng and Heintz, 2016a). It presents MSTL, which is a spatio-temporal logic that combines the well-known MTL metric temporal logic with qualitative spatial relations from RCC-8. Subsequently, efficient techniques for evaluating MSTL formulas are presented and empirically evaluated.

3.1 Introduction

Qualitative spatio-temporal reasoning is concerned with reasoning over time and space, in particular reasoning about spatial change (Cohn and Renz, 2008). This thesis presents a logic for spatio-temporal stream reasoning, alongside the tools required to incrementally evaluate spatio-temporal formulas in this logic. Using a formal logic allows us to precisely formulate conditions and constraints. Furthermore, this chapter presents techniques that allow us to efficiently determine the truth value of such a formula.

Combining spatial and temporal reasoning can be extremely useful in situations wherein one deals with for example physical objects, as it allows for the expression of spatial constraints that must hold over time. Consider the following example concerning a quad-rotor.

Example 5 (Containment in a virtual box)**.** *A quad-rotor is a small unmanned aerial vehicle that can be used in small spaces, for example indoors. In some cases, a quad-rotor may have to share space together with humans. Safety conditions could*

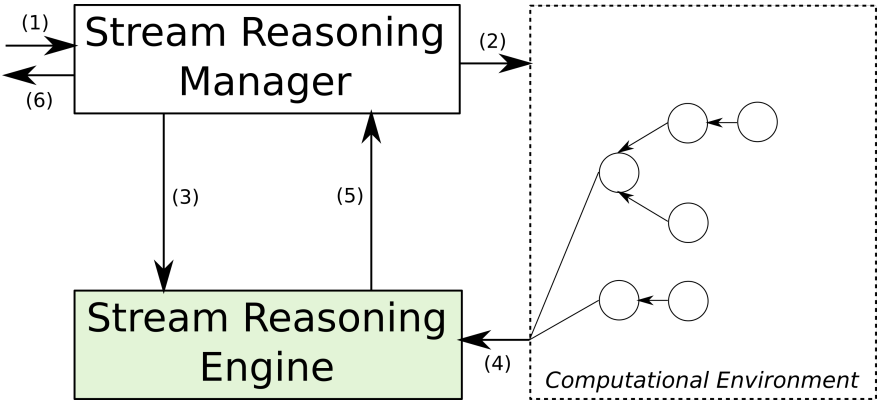

Figure 3.1: Conceptual overview with the spatio-temporal stream reasoning highlighted.

include restricting such a quad-rotor to a specific area of space, like a virtual box. An example statement combining spatial and temporal constraints is as follows: "It is always the case that if the UAV leaves the virtual box, it should be inside the virtual box within five seconds."

The constraints above are useful to detect situations where safety is compromised. A different example concerns itself with the detection of suspicious activity in order to prevent unsafe situations from occurring in the first place.

Example 6 (Perimeter monitoring)**.** *Consider a restricted area close to a public road. The area's perimeter is under surveillance by autonomous UAVs. A high-level task planner is responsible for detecting and tracking intrusions. An example rule could be expressed as: "If a moving object outside the perimeter stops moving for more than 60 seconds, dispatch a UAV to that object."*

In the above example, a type of spatio-temporal behaviour can be detected and responded to. Note that neither example deals with exact spatial coordinates. Rather, spatial entities are referenced by their spatial relations. This thesis therefore focuses on qualitative spatial relations when dealing with the spatial properties of objects.

The conceptual overview from before, with the relevant spatio-temporal stream reasoning components highlighted, is shown in Figure 3.1. In particular, this chapter focuses on the evaluation of spatio-temporal logic formulas.

3.1.1 Spatial reasoning with RCC-8

The *region connection calculus* RCC was presented by Randell et al. (1992) as a calculus for topological reasoning over abstract regions based on their

Definition	Description
$\mathsf{C}(x,y) \equiv_{def} x \cap y \neq \varnothing$	Connected
$\mathsf{DC}(x,y) \equiv_{def} \neg\mathsf{C}(x,y)$	Disconnected
$\mathsf{P}(x,y) \equiv_{def} \forall z[\mathsf{C}(z,x) \rightarrow \mathsf{C}(z,y)]$	Part of
$\mathsf{PP}(x,y) \equiv_{def} \mathsf{P}(x,y) \wedge \neg\mathsf{P}(y,x)$	Proper part
$\mathsf{EQ}(x,y) \equiv_{def} \mathsf{P}(x,y) \wedge \mathsf{P}(y,x)$	Equals
$\mathsf{O}(x,y) \equiv_{def} \exists z[\mathsf{P}(z,x) \wedge \mathsf{P}(z,y)]$	Overlapping
$\mathsf{PO}(x,y) \equiv_{def} \mathsf{O}(x,y) \wedge \neg\mathsf{P}(x,y) \wedge \neg\mathsf{P}(y,x)$	Partially overlapping
$\mathsf{DR}(x,y) \equiv_{def} \neg\mathsf{O}(x,y)$	Discrete from
$\mathsf{TPP}(x,y) \equiv_{def} \mathsf{PP}(x,y) \wedge \exists z[\mathsf{EC}(z,x) \wedge \mathsf{EC}(z,y)]$	Tangential proper part
$\mathsf{EC}(x,y) \equiv_{def} \mathsf{C}(x,y) \wedge \neg\mathsf{O}(x,y)$	Externally connected
$\mathsf{NTPP}(x,y) \equiv_{def} \mathsf{PP}(x,y)$ $\wedge\neg\exists z[\mathsf{EC}(z,x) \wedge \mathsf{EC}(z,y)]$	Non-tangential proper part
$\mathsf{P}^{-1}(x,y) \equiv_{def} \mathsf{P}(y,x)$	Inverse part of
$\mathsf{PP}^{-1}(x,y) \equiv_{def} \mathsf{PP}(y,x)$	Inverse proper part
$\mathsf{TPP}^{-1}(x,y) \equiv_{def} \mathsf{TPP}(y,x)$	Inverse tangential proper part
$\mathsf{NTPP}^{-1}(x,y) \equiv_{def} \mathsf{NTPP}(y,x)$	Inverse non-tangential proper part

Table 3.1: Definitions for the 15 RCC relations.

spatial relations. These regions are assumed to be composed of non-empty regions of topological space that can be characterised in terms of sets of points. The calculus defines and builds up spatial relations between regions from a primitive 'connected' relation $\mathsf{C}(x,y)$, which has the intended meaning that (non-empty) regions x and y share at least one point. Randell et al. (1992) recursively define a set of 15 RCC relations (including C) as shown in Table 3.1.[4]

RCC-8 is a subset of RCC that is composed of eight jointly exhaustive and pairwise disjoint relations that allow us to describe the topological spatial relations between regions. Using composition-table based reasoning in RCC-8 (Cui et al., 1993), new spatial relations can be inferred from incomplete spatial knowledge. Figure 3.2 shows the eight qualitative relations that are considered by RCC-8 as well as their transitions. The transitions are interesting in situations where observations of a pair of regions yield non-adjacent spatial relations, because those intermediate and unobserved relations can then be inferred.

Example 7 (Busy student)**.** *Suppose that we have a spatial configuration in which we consider three regions* student, office, *and* canteen. *A robot observes that region* student *is strictly within region* office, *i.e.* $\mathsf{NTTP}(\mathsf{student},\mathsf{office})$. *Further, the robot knows that region* canteen *is disconnected from region* office, *i.e.* $\mathsf{DC}(\mathsf{canteen},\mathsf{office})$. *When asked whether the student is in the canteen, the*

[4]Sometimes the relation $\mathsf{EQ}(x,y)$ is written with infix notation instead, i.e. $x = y$.

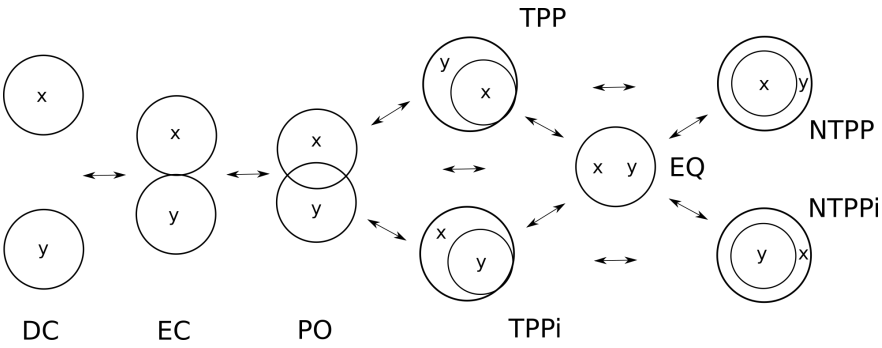

Figure 3.2: The eight qualitative spatial relations considered by RCC-8 and their transitions as illustrated by regions x and y.

robot cannot rely on direct observations. In fact, the robot might even consider it likely for a student to be in a canteen. By using the composition table for RCC-8, the robot can correctly deduce the unobserved spatial relation $\mathsf{DC}(\mathsf{student}, \mathsf{canteen})$.

In the above example, the observed spatial relations are used to infer unobserved facts about the world. This can be especially useful when there is a need for information that is not easily observable, or even unobservable.

3.1.2 Temporal reasoning with MTL

Recall that LTL formulas lacked the expressivity to describe quantitative temporal constraints such as 'within 10 seconds'. *Metric Temporal Logic* (MTL) was introduced by Koymans (1990) and solves this problem by adding temporal intervals to the $\mathcal{U}$ operator, yielding $\mathcal{U}_I$ for some temporal interval I.

Definition 7 (MTL syntax)**.** *The syntax for MTL is as follows for atomic propositions* $p \in \mathsf{Prop}$*, temporal interval* $I \subseteq (0, \infty)$*, and well-formed formulas (wffs)* ϕ *and* ψ*:*

$$p \mid \neg\phi \mid \phi \vee \psi \mid \phi \,\mathcal{U}_I\, \psi \tag{3.1}$$

As with LTL, additional Boolean operators are commonly used for MTL. These are the same as those for LTL, with the exception of the modal operators $\Box$ and $\Diamond$ which become $\Box_I$ and $\Diamond_I$. In MTL, $\Box$ and $\Diamond$ become short-hand for (unconstrained) $\Box_{[0,\infty]}$ and $\Diamond_{[0,\infty]}$ instead.

Example 8 (MTL statement)**.** *Recall the statement from a previous synergy example: "If I am in my room, it is always the case that if the light condition of my surroundings is poor, then the surroundings will be well-lit within 10 seconds." In MTL, this statement can be written as*

$$\mathsf{inRoom} \rightarrow (\Box(\mathsf{poorLight} \rightarrow \Diamond_{[0,10]}\mathsf{goodLight})), \tag{3.2}$$

where propositions inRoom, poorLight *and* goodLight *stand for "I am in my room", "the light condition of my surroundings is poor", and "the surroundings are well-lit" respectively.*

MTL thus makes it possible to be more precise about temporal intervals by making them explicit in the temporal operators. The semantics of MTL is given below.

Definition 8 (MTL semantics)**.** *The MTL statement that a wff holds in* $\mathcal{M} = \langle T, <, V \rangle$ *at time-point* $t \in T$ *is defined recursively:*

$$\mathcal{M}, t \models p \text{ iff } p \in V(t) \text{ for } p \in \mathsf{Prop} \tag{3.3}$$

$$\mathcal{M}, t \models \neg\phi \text{ iff } \mathcal{M}, t \not\models \phi \tag{3.4}$$

$$\mathcal{M}, t \models \phi \vee \psi \text{ iff } \mathcal{M}, t \models \phi \text{ or } \mathcal{M}, t \models \psi \tag{3.5}$$

$$\mathcal{M}, t \models \phi \, \mathcal{U}_{[t_1,t_2]} \, \psi \text{ iff } \exists t' \in [t + t_1, t + t_2] : \mathcal{M}, t' \models \psi \text{ and } \forall t'' \in [t, t') : \mathcal{M}, t'' \models \phi \tag{3.6}$$

One problem of MTL is that it allows for *punctuality* constraints. A punctuality constraint is one wherein the interval is punctual, i.e. a specific time-point rather than a range. Alur et al. (1996) prove that MTL is undecidable and introduce a decidable fragment of MTL called Metric Interval Temporal Logic (MITL) by excluding punctuality constraints. A survey of decidable MTL-sublogics covering SMTL, BMTL, CFMTL, MITL, and $\mathsf{MTL}_{[0,\infty]}$ is provided by Ouaknine and Worrell (2008).

3.1.3 Approaches towards spatio-temporal reasoning

Several qualitative spatio-temporal reasoning formalisms have been created by combining a spatial formalism with a temporal one. Examples are STCC (Gerevini and Nebel, 2002) and ARCC-8 (Bennett et al., 2002) which both combine RCC-8 with Allen's Interval Algebra (Allen, 1983).

The ST_i family[5] (Wolter and Zakharyaschev, 2000) of spatio-temporal logics represent a language for reasoning over spatio-temporal representations and offers such a temporalisation of RCC-8 using temporal operators similar to MTL. ST_i member language ST_0 makes use of the temporal operators 'it will always be the case' $\Box$, 'at some point in the future' $\Diamond$, and 'at the next time-point' $\bigcirc$. Its extension ST_1 introduces spatio-temporal representations for spatial relations between two time-points through the 'next' operator, but does not attempt to provide reasoning techniques that handle such instantaneous observations. One problem is for example that ST_1 can refer to future states, which clearly causes difficulties when observations are assumed to be incremental over time. Furthermore, the ST_i

[5]For consistency reasons this thesis uses the same typesetting for all logics; the original literature—as well as the papers this chapter is based on—use a calligraphic version $\mathcal{ST}_i$ instead.

family is a pure temporalisation of RCC-8 in the sense that it does not allow for expressing other (non-spatial) properties. This means that the domain of discourse exclusively treats its objects as spatial entities in relation to each other.

A survey of other approaches that combine spatial and temporal reasoning techniques is provided by Kontchakov et al. (2007).

3.2 Metric Spatio-Temporal Logic

To make statements about the spatial and temporal nature of objects, we introduce a hybrid logic called *Metric Spatio-Temporal Logic* (MSTL), which combines elements from MTL and RCC-8. MTL provides the ability to reason over objects in time, but does not include a spatial formalism. We extend these languages by considering temporal objects that are spatial in nature. MSTL is thus similar to ST_1, which temporalises RCC-8 but restricts its language to spatial relations. Because MSTL is in part based on MTL, statements in MSTL can contain both spatial relations and predicates.

3.2.1 Syntax

Spatial relations are of the form $R(r_1, r_2)$ where R is any of

$$\left\{\mathsf{EC}, \mathsf{EQ}, \mathsf{DC}, \mathsf{PO}, \mathsf{NTTP}, \mathsf{TPP}, \mathsf{NTTP}^{-1}, \mathsf{TPP}^{-1}\right\} \tag{3.7}$$

and r_1, r_2 are spatial objects, also referred to as regions. We call this set $\mathcal{R}_8$ for brevity to indicate that its elements correspond to the RCC-8 relations 'externally connected', 'equals', 'disconnected', 'non-tangential proper part', 'tangential proper part', 'inverse non-tangential proper part' and 'inverse tangential proper part' respectively. Given an n-ary predicate P, binary spatial relation $\mathcal{R}_8$, variable or constant terms $\tau_1, \ldots, \tau_n$, and integers $i, j \in \mathbb{Z}$ the following statements are well-formed formulas (wffs) in MSTL:

$$\mathcal{R}_8(\bigcirc^i \tau_1, \bigcirc^j \tau_2) \mid P(\tau_1, \ldots, \tau_n) \mid \bigcirc^i \tau_1 = \bigcirc^j \tau_2 \tag{3.8}$$

We will write τ for $\bigcirc^0 \tau$, $\bigcirc \tau$ for $\bigcirc^1 \tau$, and $\bigcirc^- \tau$ for $\bigcirc^{-1} \tau$ as syntactic sugar. By recursion, for wffs ϕ and ψ and variable x the following statements are also wffs in MSTL:

$$\neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \phi \rightarrow \psi \mid \forall x[\phi] \mid \exists x[\phi] \tag{3.9}$$

Finally, temporal notations are also defined by recursion for wff ϕ, natural numbers $n_1, n_2 \in \mathbb{N}$, and integers $i \in \mathbb{Z}$:

$$\bigcirc^i \phi \mid \Box_{[n_1,n_2]} \phi \mid \Box \phi \mid \Diamond_{[n_1,n_2]} \phi \mid \Diamond \phi \mid \phi \, \mathcal{U}_{[n_1,n_2]} \, \psi \tag{3.10}$$

Note that we apply the same syntactic sugar as for $\bigcirc$ over terms.

The syntax allows us to make complex spatio-temporal statements. Take for example the following statement, where informally $\Box$ means 'it will always be the case', $\Diamond$ means 'at some point in the future', and $\bigcirc$ means 'at the next time-point'. The spatial relation PO is contained in $\mathcal{R}_8$ and stands for 'partially overlapping'.

$$\begin{aligned} &\forall c_1[\forall c_2[c_1 \neq c_2 \wedge Car(c_1) \wedge Car(c_2) \rightarrow \\ &\quad (\Box(\mathsf{PO}(\bigcirc c_1, c_2) \wedge Speeding(c_1) \rightarrow \Diamond \mathsf{PO}(c_1, c_2)))]] \end{aligned} \tag{3.11}$$

This wff has the intended meaning 'it is always the case that if a car is speeding and tails another car, they will eventually collide'.

3.2.2 Semantics

Because we are interested in statements over space and time, we make use of *spatio-temporal models* for MSTL. It borrows the notion of a spatial assignment function from the *topological temporal model* (tt-model) from ST_i.

Definition 9 (Spatio-temporal model)**.** *A* spatio-temporal model *is a tuple of the form $\mathcal{M} = \langle T, <, U, \mathcal{D}, I, \alpha \rangle$, where T represents a set of time-points, $<$ represents an ordering over T, U represents the non-empty universe of the space as a set of points, and $\mathcal{D} = \langle \mathcal{P}, \mathcal{R} \rangle$ represents the domain consisting of predicates $\mathcal{P}$ and spatial objects $\mathcal{R}$. An interpretation $I^t \in I$ maps predicates and constant terms to $\mathcal{P}$ and $\mathcal{R}$ respectively for every time-point in T. For constant terms this mapping will be the same for all t, but for predicates this is not necessarily the case. A spatial assignment function α associates at every time-point in T every spatial object label in $\mathcal{R}$ to a subset of U. It is extended to interpret 'next' as $\alpha(\bigcirc^i r, t) = \alpha(\bigcirc^{i-j} r, t + j)$ for spatial object label $r \in \mathcal{R}$ and integers $i, j \in \mathbb{Z}$.*

From this definition it is clear that we are only considering objects that have some spatial properties associated with them, expressed in the form of spatial relations. Spatial objects therefore are also commonly called *regions* when we only focus on temporal and spatial properties.

Definition 10 (Truth)**.** *The MSTL statement that a spatio-temporal formula ϕ holds in $\mathcal{M} = \langle T, <, U, \mathcal{D}, I, \alpha \rangle$ at time-point $t \in T$ is defined recursively for*

integers $i, j \in \mathbb{Z}$.

$$\mathcal{M}, t \models P(\tau_1, \ldots, \tau_n) \text{ iff } \langle I^t(\tau_1), \ldots, I^t(\tau_n) \rangle \in I^t(P) \tag{3.12}$$

$$\mathcal{M}, t \models \forall x[\phi] \text{ iff } \forall r \in \mathcal{R} : \mathcal{M}, t \models \phi[x/r] \tag{3.13}$$

$$\mathcal{M}, t \models \exists x[\phi] \text{ iff } \exists r \in \mathcal{R} : \mathcal{M}, t \models \phi[x/r] \tag{3.14}$$

$$\mathcal{M}, t \models \neg\phi \text{ iff } \mathcal{M}, t \not\models \phi \tag{3.15}$$

$$\mathcal{M}, t \models \phi \vee \psi \text{ iff } \mathcal{M}, t \models \phi \text{ or } \mathcal{M}, t \models \psi \tag{3.16}$$

$$\mathcal{M}, t \models \phi \, \mathcal{U}_{[t_1,t_2]} \, \psi \text{ iff } \exists t' \in [t + t_1, t + t_2] : \mathcal{M}, t' \models \psi \text{ and } \forall t'' \in [t, t') : \mathcal{M}, t'' \models \phi \tag{3.17}$$

$$\mathcal{M}, t \models \bigcirc^i \phi \text{ iff } \mathcal{M}, t + i \models \phi \tag{3.18}$$

$$\mathcal{M}, t \models \mathsf{C}(\bigcirc^i r_1, \bigcirc^j r_2) \text{ iff } \alpha(r_1, t + i) \cap \alpha(r_2, t + j) \neq \emptyset \tag{3.19}$$

From the RCC 'connected' spatial relation C, the usual semantics of all RCC-8 relations can be recursively defined, but here they are left out for the sake of brevity.

Allowing for the 'next' operator to be invoked over region variables is a powerful extension that makes it possible to refer to a particular region at the next time-point, or by recursive application any past or future time-point.

3.3 Spatio-temporal inference with RCC-8

RCC-8 allows for both representation of observed spatial relations as well as the inference of unobserved spatial relations. However, these observations are usually assumed to be restricted to a single time-point rather than across different time-points. To represent spatial relations across time-points, we can add a temporal element. The addition of a 'next' operator $\bigcirc$ as initially proposed by ST_1 can lead to situations wherein regions at different time-points are considered. In what follows, we explore the consequences to spatio-temporal inference when the 'next' operator is used to describe relations across time-points, starting with the representation of these relations.

3.3.1 Temporal constraint networks

While the 'next' operator allows for powerful representations, it complicates evaluation of those statements when we consider observations of the world to occur within rather than across time-points. Spatial relations for regions can be partially observed at time-point t and at time-point $t + 1$ independently, but no observations can be made with regards to the spatial relations between regions at time-point t and regions at time-point $t + 1$. To better illustrate how these concepts relate, we introduce the spatial relation matrix as a representation of constraint networks.

Definition 11 (Spatial relation matrix)**.** *Given a spatio-temporal model $\mathcal{M}$, a* spatial relation matrix *is an $n \times n$ matrix M^t for time-point $t \in T$ where n denotes the total number of region variables $|\mathcal{R}|$. For every matrix element $M^t_{i,j}$ and region variables $r_i, r_j \in \mathcal{R}$ we have $M^t_{i,j} = (r_i R r_j)$ such that $R \subseteq \mathcal{R}_8$ and $R \neq \emptyset$. The semantics of M^t are then as follows.*

$$M^t_{i,j} = (r_i R r_j) \text{ iff } \mathcal{M}, t \models \bigvee_{R_k \in R} R_k(r_i, r_j) \tag{3.20}$$

The spatial relation matrix allows us to intuitively represent spatial facts about regions and corresponds to a complete RCC-8 network. Such matrices are also expected as input to qualitative reasoners such as GQR. The main diagonal always consists of the singleton $\{\mathsf{EQ}\}$. Further, the matrix is semi-symmetric; symmetry holds for all relations except for NTTP and TPP, which have inverses NTTP^{-1} and TPP^{-1} respectively. Existing general solvers for qualitative CSPs such as GQR can be used to determine the algebraic closure of spatial relation matrices, i.e. given spatial relation matrix M^t, the algebraic closure $AC(M^t)$ yields a spatial relation matrix N^t such that for every corresponding set of spatial relations $N^t_{i,j} \subseteq M^t_{i,j} \subseteq \mathcal{R}_8$. A small example of a spatial relation matrix for regions r_1, r_2, r_3 at time-point t with partial knowledge is shown below.

$$M^t = \begin{bmatrix} \{\mathsf{EQ}\} & \left\{\mathsf{NTTP}^{-1}\right\} & \{\mathsf{PO}, \mathsf{EC}\} \\ \{\mathsf{NTTP}\} & \{\mathsf{EQ}\} & \{\mathsf{DC}\} \\ \{\mathsf{PO}, \mathsf{EC}\} & \{\mathsf{DC}\} & \{\mathsf{EQ}\} \end{bmatrix} \tag{3.21}$$

Region r_2 is inside of region r_1 but disconnected from region r_3, and region r_1 is partially overlapping or externally connected with region r_3.

A spatial relation matrix can be extended to describe relations between multiple time-points. This is a useful property because it allows us to describe relations between regions at different time-points that are not necessarily consecutive.

Definition 12 (Intertemporal spatial relation matrix)**.** *An* intertemporal spatial relation matrix *M^{t_1,t_2} is a spatial relation matrix describing the spatial relations between regions $r_i, r_j \in \mathcal{R}$ such that we relate r_i at time-point t_1 to r_j at time-point t_2, i.e. relating $\alpha(r_i, t_1)$ to $\alpha(r_j, t_2)$.*

A spatial relation matrix M^t from Definition 11 is then equivalent to an intertemporal spatial relation matrix $M^{t,t}$. Intertemporal spatial relations can thus be represented by an intertemporal spatial relation matrix. For the 'next' operator, this would for example be $M^{t,t+1}$. However, we assume that these relations are unobservable and must somehow be inferred from our observations at time-points t and $t+1$, represented by M^t and M^{t+1}.

By combining the four different combinations for intertemporal spatial relation matrices over two time-points t_1 and t_2, we can concisely describe in one matrix the relations between regions at single time-points as well as the relations between those regions at different time-points. This corresponds to an RCC-8 network in which every region is contained twice, i.e. once for every time-point.

Definition 13 (Extended spatial relation matrix)**.** *An* extended spatial relation matrix $M^{t_1 \cup t_2}$ *for* $t_1 < t_2$ *combines four intertemporal spatial relation matrices as follows:*

$$M^{t_1 \cup t_2} = \begin{bmatrix} M^{t_1,t_1} & M^{t_1,t_2} \\ M^{t_2,t_1} & M^{t_2,t_2} \end{bmatrix} \tag{3.22}$$

In general, spatial relation matrices can be used to represent uncertainty for spatial relations between regions by using non-singleton sets. This is important because often we can not deduce that a single relation must hold. We can use extended spatial relation matrices to talk about the spatial relations both within individual time-points and between time-points. This makes them a suitable representation tool for intertemporal RCC-8 networks when considering the problem of deducing unobservable intertemporal relations.

3.3.2 Intratemporal inference

Intratemporal inference with RCC-8 assumes that all spatial relations are observed within the same time-point, i.e. $M^{t,t}$ for some time-point t. In this case, $M^{t,t}$ represents a constraint network for a single time-point, for which it may be possible to reduce the uncertainty of spatial relations between regions based on the observed spatial relations between other regions. It is possible to apply composition table based reasoning for RCC-8 to this effect. A composition table presumes regions i, j, and k such that the spatial relations for (i, j) and (j, k) are knowns, and presents the possible spatial relations that may exist between regions (i, k).

Gantner et al. (2008) present the *Generic Qualitative Solver* (GQR) which can be used to perform qualitative reasoning on a number of calculi, including RCC-8. They make use of the path consistency algorithm shown in Algorithm 1, based on the path consistency algorithm by Mackworth (1977). The algorithm takes a constraint network and produces a refined constraint network in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space. Path consistency continuously updates spatial relation C_{ik} by computing $C_{ik} \cap (C_{ij} \circ C_{jk})$, utilising a third variable j. These updates can be performed based on a composition table.

3.3.3 Intertemporal inference

Sometimes we want to talk about spatial relations between regions at different time-points. By following the example of ST_1, we can extend our

Algorithm 1: Path consistency (Gantner et al., 2008)

```
1  function PATH-CONSISTENCY((V, C)):
2  Q ← {(i, j) | 1 ≤ i < j ≤ n}
3  while Q is not empty do
4      select and delete an (i, j) from Q
5      for k ← 2 to n, k ≠ i ∧ k ≠ j do
6          t ← C_ik ∩ (C_ij ∘ C_jk)
7          if t ≠ C_ik then
8              C_ik ← t
9              C_ki ← t⌣
10             Q ← Q ∪ {(i, j)}
11         end
12         t ← C_kj ∩ (C_ki ∘ C_ij)
13         if t ≠ C_kj then
14             C_kj ← t
15             C_jk ← t⌣
16             Q ← Q ∪ {(k, j)}
17         end
18     end
19 end
20 return (V, C)
```

definition of region symbols accordingly. If region is a region symbol, then ◯region is also a region symbol, such that $\alpha(\bigcirc\text{region}, t) = \alpha(\text{region}, t+1)$. This allows us to refer to regions at different time-points using the same region symbol region. However, this also complicates the semantics of the mapping α. From its definition, it is clear that we are referring to the same universe of points, but it is not clear whether $\alpha(x,t) = \alpha(x,t+1)$ for all time-point t, or whether it is possible that $\alpha(x,t) \neq \alpha(x,t+1)$ for some time-point t. This leads to two opposing interpretations of space.

Rigid regions In the case of *rigid regions*, the points that are associated with a region symbol are the same across all time-points, i.e.

$$\forall t \in T[x \in \mathcal{R} \rightarrow \alpha(x,t) = \alpha(x,t+1)] \tag{3.23}$$

As an example, consider a set of points that make up the region of a cookie, which we refer to using the region symbol cookie. At time-point t_1 we then assume that $\alpha(\text{cookie}, t_1) = R_{cookie}$ such that $R_{cookie} \subseteq U$. If we at the next time-point t_2 break the cookie into two parts, indicated by regions R_{left} and R_{right}, it is clearly the case that $\alpha(\bigcirc\text{cookie}, t_1) = R_{left} \cup R_{right}$ for a non-crumbling cookie. Thus the relation between the cookie and its individual components it preserved across time, which is a nice property.

The trade-off is however that the region symbols are always mapped to the same set of points in U; there is no difference between a region symbol

and a region. Any intertemporal relations between the same region will therefore always correspond to $\mathsf{EQ}(x, \bigcirc x)$. This makes sense if we want to check whether the region referred to by region symbol x has itself changed point-wise, but it is less useful if we want to know whether a cookie's relation to space has somehow changed. Consider a second region R_{table} with symbol table. Suppose that we first observe that the cookie is within the area of the table, i.e. $\mathsf{NTPP}(\mathsf{cookie}, \mathsf{table})$ at time-point t_1, and then observe at time-point t_2 that $\mathsf{DC}(\mathsf{cookie}, \mathsf{table})$. This means that at time-point t_1, all points in R_{cookie} are also part of R_{table} minus the tangential points; $R_{cookie} \subset R_{table}$. At time-point t_2, we have $R_{cookie} \cap R_{table} = \varnothing$. Since the rigid region assumption states that $\alpha(x, t) = \alpha(x, t+1)$, we now have a problem, as cookie cannot both be separate from and a subset of table. The object space assumption is therefore unsound for intertemporal relations.

Rigid space A different way of dealing with intertemporal relations is by using the *rigid space* assumption, where we do not require all region symbols to refer to the same region at all time-points. This means that the region symbol cookie is α-mapped to different sets of spatial points at different time-points. Suppose $\alpha(\mathsf{cookie}, t) = R_{cookie}$ and $\alpha(\bigcirc\mathsf{cookie}, t) = R'_{cookie}$, then it is possible that $R_{cookie} \neq R'_{cookie}$. One could see this distinction as the cookie occupying a different area of space, and that both the original and new areas of occupied space have remained the same in terms of points. However, without any context that means we cannot say anything about intertemporal relations, although the inferencing power for intratemporal relations remain unchanged.

The rigid space assumption therefore assumes space itself is rigid. We can now use space itself as a frame of reference for intertemporal spatial relations. Recall the cookie and the table, where $\mathsf{NTPP}(\mathsf{cookie}, \mathsf{table})$ and $\mathsf{DC}(\bigcirc\mathsf{cookie}, \bigcirc\mathsf{table})$ This is no longer a problem; it simply means that

$$\alpha(\mathsf{cookie}, t_1) \neq \alpha(\mathsf{cookie}, t_2) \vee \alpha(\mathsf{table}, t_1) \neq \alpha(\mathsf{table}, t_2), \tag{3.24}$$

i.e. either the cookie moved, or the table moved, or both moved. In this thesis we make use of the rigid space assumption when referring to intertemporal relations.

Reasoning alone thus does not allow us to say anything about intertemporal relations, represented by M^{t_1,t_2} and M^{t_2,t_1} in extended spatial relation matrices. These relations cannot be observed, nor can they be inferred from individual time-points. Concretely, observations are limited to M^{t_1,t_1} and M^{t_2,t_2}. This may seem counter-intuitive, but this is because humans often assume a *frame of reference* when observing spatial changes over time. One way around this problem is therefore to make assumptions about some or all intertemporal relations represented by M^{t_1,t_2} and M^{t_2,t_1} in order to establish such a frame of reference. Effectively this corresponds to 'pegging'

only these landmark regions to the space they occupy, allowing outside space to warp relative to the landmarks and fixing the frame of reference. The set of landmarks is indicated by $\mathcal{LM} \subseteq \mathcal{R}$. For all landmarks $x \in \mathcal{LM}$, the α-mapping is fixed such that $\alpha(x,t) = \alpha(x,t+1)$ for all $t \in T$. By using a consistent set of landmarks, it is possible to infer intertemporal relations based on the spatial relations between non-landmark and landmark regions. Additionally, since the landmark regions are rigid, the spatial relations between landmark regions do not change.

Definition 14 (Landmark)**.** *A* landmark *given a set of region variables $\mathcal{R}$ over any two time-points $t, t+1$ is a region variable $r \in \mathcal{R}$ that is* rigid *between t and $t+1$, i.e. $\mathsf{EQ}(r, \bigcirc r)$. The set of landmarks is indicated by $\mathcal{LM} \subseteq \mathcal{R}$ such that $r \in \mathcal{LM}$ implies that landmark r is rigid.*

Example real-world landmark candidates are e.g. buildings, lakes, monuments, trees, and roads. These physical entities are unlikely to change during the run-time of a system, and therefore provide a reasonable frame of reference. An immediate effect of landmarks being rigid is that their relations to other landmark regions remain unchanged. Effectively, the set of landmarks $\mathcal{LM}$ provides a possible frame of reference with respect to which relations may change over time. Since this affects the truth semantics of statements in MSTL, we introduce a landmark extension to the spatio-temporal model to capture this.

Definition 15 (Landmark-based spatio-temporal model)**.** *A* landmark-based spatio-temporal model *is a spatio-temporal model*

$$\mathcal{M}_{\mathcal{LM}} = \langle T, <, U, \mathcal{D}, I, \alpha_{\mathcal{LM}} \rangle \tag{3.25}$$

and $\mathcal{LM} \subseteq \mathcal{R}$ represents the landmark set. $\mathcal{LM}$ then restricts α such that for all time-points $t \in T$ and all landmark regions $r \in \mathcal{LM}$ it is the case that $\alpha(r,t) = \alpha(r,t+1)$.

Landmarks may introduce inconsistencies if we make observations that conflict with the landmark-imposed restriction of α. To illustrate how this might happen, consider an example where at time-point t we make the observation $\mathsf{PO}(r_1, r_2)$, and at time-point $t+1$ we make the observation $\mathsf{DC}(r_1, r_2)$. If we only consider the individual time-points, there is no problem. The following extended spatial relation matrix illustrates our ignorance of the intertemporal spatial relations M^{t_1,t_2} and M^{t_2,t_1}.

$$M^{t_1 \cup t_2} = \begin{bmatrix} \{EQ\} & \{PO\} & \mathcal{R}_8 & \mathcal{R}_8 \\ \{PO\} & \{EQ\} & \mathcal{R}_8 & \mathcal{R}_8 \\ \mathcal{R}_8 & \mathcal{R}_8 & \{EQ\} & \{DC\} \\ \mathcal{R}_8 & \mathcal{R}_8 & \{DC\} & \{EQ\} \end{bmatrix} \tag{3.26}$$

However, if we use landmarks, the choice of $\mathcal{LM}$ results in an assumption about some intertemporal relations. Choosing $\mathcal{LM} = \{r_1, r_2\}$ is inconsistent, because it implies that regions r_1 and r_2 need to be partially overlapping and disconnected at the same time, which is a contradiction. Instead

picking $\mathcal{LM} = \{r_1\}$ is consistent, and one could imagine region r_2 'moving away from' region r_1. Naturally, the converse holds as well if we pick region r_2 as our frame of reference.

We can show that consistency is guaranteed if only one landmark is chosen, and the above example shows that this does not always hold for the case of $|\mathcal{LM}| \geq 2$. Picking a single landmark corresponds to the case of adding a single connection between two disconnected RCC-8 networks for different time-points. The issue of choosing more than one landmark while retaining consistency is a difficult problem, and is closely related to the Amalgamation Property (Li et al., 2008), as well as the Patchwork Property (Lutz and Miličić, 2007; Huang, 2012).

To further illustrate the impact of the choice of $\mathcal{LM}$, consider again the scenario above and suppose we wish to evaluate the formula $\Box\mathsf{EQ}(r_1, \bigcirc r_1)$ at time-point t. Choosing $\mathcal{LM} = \{r_1\}$ means this formula will evaluate to True, i.e.

$$\mathcal{M}_{\{r_1\}}, t \models \Box\mathsf{EQ}(r_1, \bigcirc r_1). \tag{3.27}$$

Choosing $\mathcal{LM} = \{r_2\}$ means this formula will evaluate to False, i.e.

$$\mathcal{M}_{\{r_2\}}, t \not\models \Box\mathsf{EQ}(r_1, \bigcirc r_1). \tag{3.28}$$

Choosing any other consistent $\mathcal{LM}$ we can only conclude

$$\mathcal{M}_{\mathcal{LM}}, t \models \Box\mathsf{EQ}(r_1, \bigcirc r_1) \vee \neg(\Box\mathsf{EQ}(r_1, \bigcirc r_1)); \tag{3.29}$$

we cannot say for certain which one is true. This is specifically caused by the choice of landmark in combination with the observations at the two time-points. The following two statements then hold for the same two observations described earlier:

$$\mathcal{M}_{\{r_1\}}, t \models \Box\mathsf{EQ}(r_1, \bigcirc r_1) \wedge \neg\Box\mathsf{EQ}(r_2, \bigcirc r_2) \tag{3.30}$$

$$\mathcal{M}_{\{r_2\}}, t \models \Box\mathsf{EQ}(r_2, \bigcirc r_2) \wedge \neg\Box\mathsf{EQ}(r_1, \bigcirc r_1) \tag{3.31}$$

This clearly shows how landmark choice shapes the frame of reference within which MSTL statements may hold.

3.4 Stream reasoning with MSTL

In stream reasoning, information is assumed to become incrementally available. Recall that progression is a technique for evaluating temporal logic formulas where we try to determine the truth value of the formula based on the information received thus far. This makes it possible to sometimes determine the truth value for an MSTL formula without having to wait for the entire stream to arrive. The result of progressing a formula through the first state in a sequence is a new formula that holds in the remainder of the state sequence iff the original formula holds in the complete state sequence. If progression returns true (false), the entire formula must be true (false), regardless of future states.

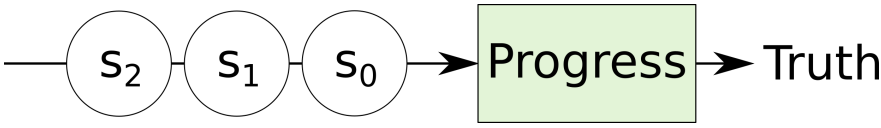

Figure 3.3: Conceptual representation of progression.

3.4.1 Progression of MTL

In the area of model checking, given a Kripke model $\mathcal{M}$ and a wff ϕ, the task is to determine all states s such that $\mathcal{M}, s \models \phi$. Progression instead is tasked with the problem of incrementally determining whether $\mathcal{M}, s \models \phi$ by iterating over the temporal states in $\mathcal{M}$. Progression thus nicely fits the context of stream reasoning wherein streams become available incrementally. This is shown conceptually in Figure 3.3, where states s_i are processed by the progression algorithm in sequence.

The progression algorithm originally appeared in Bacchus and Kabanza (1996, 1998) for MTL and is reproduced as Algorithm 2. It takes a wff ϕ, state s_i, and a time duration Δ until the next successor state s_{i+1}. It then produces a rewritten formula ϕ^+ that incorporates the state information contained within s_i. Note that the algorithm does not assume access to $\mathcal{M}$; instead, individual states are considered. If $\texttt{PROGRESS}(\phi, s_i, \Delta, \mathcal{D})$ yields ϕ^+, then the next call will be to $\texttt{PROGRESS}(\phi^+, s_{i+1}, \Delta, \mathcal{D})$ etc. If $\texttt{PROGRESS}(\phi, s_i, \Delta, \mathcal{D})$ returns True, then we can conclude $\mathcal{M}, s_i \models \phi$; and $\mathcal{M}, s_i \not\models \phi$ otherwise. Bacchus and Kabanza (1998) show the correctness of PROGRESS by proving that

$$\mathcal{M}, s_i \models \phi \text{ iff } \mathcal{M}, s_{i+1} \models \texttt{PROGRESS}(\phi, s_i, \Delta, \mathcal{D}) \tag{3.32}$$

for any state s_i, where $\Delta = \mathcal{T}(s_{i+1}) - \mathcal{T}(s_i)$ denotes the time between the states s_i and s_{i+1}. The complexity of progression is linear in the size of the formula, but the resulting formula may double in size. This may result in exponentially long formulas in the worst case, but by introducing intervals for temporal operators, the worst-case length can be limited.

3.4.2 Spatial state streams

Progression works by taking state information and rewriting the formula based on the received state. Formulas are then evaluated based on a sequence of states, each of which syntactically rewrites the formula until a truth value can be determined. Such sequences of states are called *state streams*. State streams are generated by an underlying system, often based on a combination of sensor readings and conversions in robotic applications. From the perspective of the logic, the state stream is simply a (partially observed) Kripke model over which formulas are evaluated.

To evaluate a logic formula through progression, a state must at least contain the truth value of the predicates or relations that occur in said for-

Algorithm 2: Progression for MTL (Bacchus and Kabanza, 1996)

function PROGRESS$(\phi, s_i, \Delta, \mathcal{D})$:
  if $\phi = \phi_1 \wedge \phi_2$ **then**
    return PROGRESS$(\phi_1, s_i, \Delta, \mathcal{D}) \wedge$ PROGRESS$(\phi_2, s_i, \Delta, \mathcal{D})$
  else if $\phi = \neg\phi_1$ **then**
    return $\neg$PROGRESS$(\phi_1, s_i, \Delta, \mathcal{D})$
  else if $\phi = \phi_1 \, \mathcal{U}_I \, \phi_2$ **then**
    if $I < 0$ **then**
      return False
    else if $0 \in I$ **then**
      return
        PROGRESS$(\phi_2, s_i, \Delta, \mathcal{D}) \vee$ (PROGRESS$(\phi_1, s_i, \Delta, \mathcal{D}) \wedge \phi_1 \, \mathcal{U}_{I-\Delta} \, \phi_2$)
    else
      return PROGRESS$(\phi_1, s_i, \Delta, \mathcal{D}) \wedge \phi_1 \, \mathcal{U}_{I-\Delta} \, \phi_2$
    end
  else if $\phi = \forall x[\phi_1]$ **then**
    return $\bigwedge_{c \in \mathcal{D}}$ PROGRESS$(\phi_1(x/c), s_i, \Delta, \mathcal{D})$
  else if $\phi = \exists x[\phi_1]$ **then**
    return $\bigvee_{c \in \mathcal{D}}$ PROGRESS$(\phi_1(x/c), s_i, \Delta, \mathcal{D})$
  else if $\phi = \bigcirc\phi_1$ **then**
    return ϕ_1
  else
    if $s_i \models \phi$ **then**
      return True
    else
      return False
    end
  end

mula. Such a state stream generation method is discussed in more detail in Chapter 4. Assuming there exists a method for generating streams of states and a qualitative spatial reasoner (QSR), augmenting these state streams with spatial information can be done in a number of ways. A straightforward and naive method would be to collect the complete set of spatial information for a given time-point, run it through the QSR to infer more information on the spatial relations, and then augment the state stream with these resulting spatial relations. A slightly better way would be to only augment the state stream with those spatial relations that are relevant to the stream reasoning engine. To efficiently infer implicit spatial relations we use the facts that relations between (rigid) variables that have not changed are the same and the algebraic closure for the same set of variables must be computed many times (every time some of the variables have changed). As an example, the spatial relations between static buildings do not change, so it is not necessary to compute their spatial relations at every time-point even if they are not explicitly given. If the set of variables is partitioned

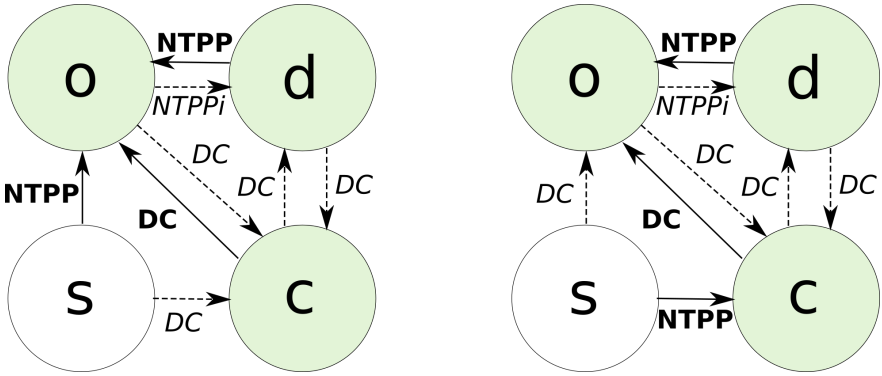

Figure 3.4: The 'busy student' scenario where regions in V_s are shaded, regions in V_d are transparent, and inferred relations are represented by dashed arrows.

into those that are static and those that are dynamic, it is enough to compute the algebraic closure of the constraints involving only static variables once and then add the constraints involving at least one changing variable when they have changed and compute the new algebraic closure. The effect is that there is an initial cost of computing the static part while the cost for each update is reduced (Heintz and de Leng, 2014).

Formally, let V^t be the set of all variables at time t and C^t be the set of all (binary) constraints on these variables at time t. The set V^t is partitioned into V_s^t and V_d^t, where V_s^t is the set of static variables and V_d^t is the set of dynamic variables at time t. C^t is partitioned into C_s^t and C_d^t, where C_s^t is the set of constraints where both variables belong to the set V_s^t and C_d^t is the set of constraints where at least one variable belong to the set V_d^t. Further, let AC_s^t denote the algebraic closure of the variables V_s^t and the constraints C_s^t and AC^t denote the algebraic closure of the variables V^t and the constraints C^t. Then, AC^t can be computed from AC_s^t by adding the constraints C_d^t and computing the algebraic closure.

Example 9 (Busy student; continued.)**.** *Recall that region* student *is strictly within region* office*, i.e.* $\mathsf{NTTP}(\mathsf{student}, \mathsf{office})$*, and region* canteen *is disconnected from region* office*, i.e.* $\mathsf{DC}(\mathsf{canteen}, \mathsf{office})$*. We can add another spatial relation; the office contains a desk* $\mathsf{NTPP}(\mathsf{desk}, \mathsf{office})$*. If the office, canteen and desk are considered to be static, i.e.* $V_s^t = \{\mathsf{office}; \mathsf{canteen}; \mathsf{desk}\}$*, we can apply preprocessing to infer* $\mathsf{DC}(\mathsf{chair}, \mathsf{canteen})$*. In contrast, since* $V_d^t = \{\mathsf{student}\}$*, we must reconsider the spatial relations involving the student at every time-point. Figure 3.4 shows how relations involving the student can change over time, while the relations between static regions remain the same.*

Figure 3.5 shows a graphical representation of evaluating the spatio-temporal formula $\Box(\mathsf{PO}(a, b) \rightarrow \Diamond\mathsf{DC}(a, b))$ given the static region vari-

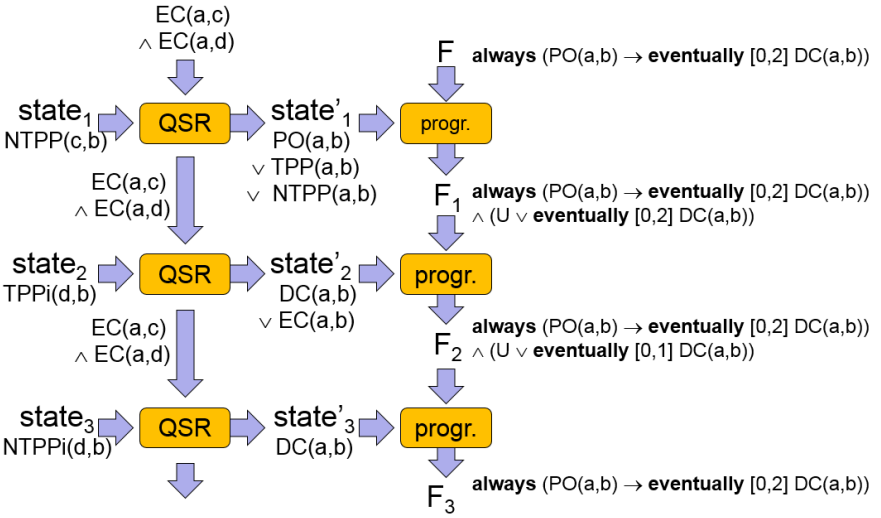

Figure 3.5: A qualitative spatio-temporal stream reasoning example.

ables a, c, d and the static relations $\mathsf{EC}(a, c)$ and $\mathsf{EC}(a, d)$. The spatial information in the first state is the spatial relation $\mathsf{NTPP}(c, b)$ which after spatial reasoning gives that $\mathsf{PO}(a, b)$ can be either True or False, meaning its truth value is Unknown. In the next state the spatial relation $\mathsf{TPP}^{-1}(d, b)$ is given from which the conclusion that $\mathsf{DC}(a, b)$ is Unknown can be drawn. Finally, in the third state where $\mathsf{NTPP}^{-1}(d, b)$ is given spatial reasoning concludes that $\mathsf{DC}(a, b)$ is True. This example shows both the benefit of spatial reasoning—as no explicit information about the relation between a and b is given—and the use of three-valued progression over disjunctive information.

3.4.3 Rewriting rules for 'next'

In order for progression to be applicable to MSTL, some changes are needed to deal with the spatial relations. In particular, the application of temporal operators to spatial objects needs to be handled before progression can operate on the propositions in a wff.

By combining temporal with spatial reasoning, we effectively need both temporal and spatial evaluation methods. Progression is used to handle temporal aspects across time-points, and has previously been used to evaluate MTL formulas (Doherty et al., 2009). For every step in the progression, spatial reasoning is performed within that step by using for example GQR. This however does not include spatial reasoning between different time-points. Therefore, progression needs to be extended to handle intertemporal relations that are the result of the 'next' operator in MSTL. This gives rise

to additional rewriting rules based on occurrences of the 'next' operator.

Progressing the 'next' operator when it occurs in front of wffs in MSTL corresponds to rewriting that formula by removing the operator, i.e. during progression $\bigcirc\phi$ is rewritten to ϕ for wff ϕ. The following proofs show equivalences for occurrence of 'next' excluding intertemporal relations, and make use of the semantics presented in Definition 10.

Theorem 1 (Next and negation)**.**

$$\models \forall x[\forall y[\neg \bigcirc R(x,y) \leftrightarrow \bigcirc \neg R(x,y)]] \tag{3.33}$$

Proof. Decomposing bi-implication into cases:

($\Rightarrow$) Assume $\mathcal{M}, t \models \neg \bigcirc R(x,y)$ holds for some arbitrary $\mathcal{M}$ and t. From the semantics of negation this means $\mathcal{M}, t \not\models \bigcirc R(x,y)$. According to the semantics of $\bigcirc$, this is equivalent to $\mathcal{M}, t+1 \not\models R(x,y)$, thus $\mathcal{M}, t+1 \models \neg R(x,y)$. Reintroducing $\bigcirc$ then yields $\mathcal{M}, t \models \bigcirc \neg R(x,y)$.

($\Leftarrow$) Analogous to the above in reverse order. □

Theorem 2 (Next and always)**.**

$$\models \forall x[\forall y[\Box_{[t_1,t_2]} \bigcirc R(x,y) \leftrightarrow \Box_{[suc(t_1),suc(t_2)]} R(x,y)]] \tag{3.34}$$

Proof. Decomposing bi-implication into cases:

($\Rightarrow$) Assume $\mathcal{M}, t \models \Box_{[t_1,t_2]} \bigcirc R(x,y)$ holds for some arbitrary $\mathcal{M}$ and t. From the semantics of $\Box$, this means $\forall t_1 \leq t' \leq t_2 : \mathcal{M}, t' \models \bigcirc R(x,y)$ holds. By definition of $\bigcirc$, for every t' we get $\mathcal{M}, suc(t') \models R(x,y)$. Reintroducing the universal quantifier, we get $\forall suc(t_1) \leq t' \leq suc(t_2) : \mathcal{M}, t' \models R(x,y)$. Reintroducing $\Box$, this yields $\mathcal{M}, t' \models \Box_{[suc(t_1),suc(t_2)]} R(x,y)$.

($\Leftarrow$) Analogous to the above in reverse order. □

Theorem 3 (Next and eventually)**.**

$$\models \forall x[\forall y[\Diamond_{[t_1,t_2]} \bigcirc R(x,y) \leftrightarrow \Diamond_{[suc(t_1),suc(t_2)]} R(x,y)]] \tag{3.35}$$

Proof. Analogous to the proof of Theorem 2, replacing symbols $\forall$ and $\Box$ by $\exists$ and $\Diamond$ respectively. □

The 'next' operator can also occur inside intertemporal relations $R(x, \bigcirc y)$. In this case, it is not possible to evaluate $R(x, \bigcirc y)$ at the current time-point, because the relation depends on a future state of y. To work around this problem, we make use of the 'previous' operator $\bigcirc^-$, which is the inverse of the 'next' operator. The following proofs show equivalences for 'next' involving intertemporal relations, and make use of the 'previous' operator.

Theorem 4 (Extract next)**.**

$$\models \forall x[\forall y[\bigcirc R(x,y) \leftrightarrow R(\bigcirc x, \bigcirc y)]] \tag{3.36}$$

Proof. Decomposing bi-implication into cases:

($\Rightarrow$) Assume $\mathcal{M}, t \models \bigcirc R(x,y)$ holds for some arbitrary $\mathcal{M}$ and t. From the semantics of $\bigcirc$, this means $\mathcal{M}, t+1 \models R(x,y)$. Further, we have $\alpha(z, t+1) = \alpha(\bigcirc z, t)$ for any region z, so we get $\mathcal{M}, t \models R(\bigcirc x, \bigcirc y)$.

($\Leftarrow$) Analogous to the above in reverse order. □

Theorem 5 (Partially extract next)**.**

$$\models \forall x[\forall y[R(x, \bigcirc y) \leftrightarrow \bigcirc R(\bigcirc^{-} x, y)]] \tag{3.37}$$

Proof. Decomposing bi-implication into cases:

($\Rightarrow$) Assume $\mathcal{M}, t \models R(x, \bigcirc y)$ holds for some arbitrary $\mathcal{M}$ and t. From the semantics of $\bigcirc$ over regions, we have $\alpha(z,t) = \alpha(\bigcirc^{-} z, t+1)$ and $\alpha(\bigcirc z, t) = \alpha(z, t+1)$ for any region z. Therefore this is equivalent to

$$\mathcal{M}, t+1 \models R(\bigcirc^{-} x, y) \tag{3.38}$$

when applied to regions x and y respectively. Introducing $\bigcirc$ then yields

$$\mathcal{M}, t \models \bigcirc R(\bigcirc^{-} x, y). \tag{3.39}$$

($\Leftarrow$) Analogous to the above in reverse order. □

The ability to rewrite MSTL formulas such that occurrences of 'next' over regions are either removed or replaced by 'previous' is vital for stream reasoning, because it allows for the delayed evaluation of formulas so that, at the time of evaluation, they only refer to the current and previous state(s) of the world. This makes the earlier-presented landmark approach applicable in a stream reasoning context.

3.4.4 Extending progression to MSTL

The progression algorithm for MTL was extended to work with MSTL. The modified algorithm is shown in Algorithm 3. It takes as its arguments a wff ϕ, time-point t, mapping α, duration Δ and domain of discourse $\mathcal{D}$ and returns a progressed formula ϕ^{+}. The duration Δ determines the metric distance between two consecutive time-points.

One limitation of the algorithm is that it requires full knowledge of the topological space through the α mapping, which in practice is not directly

Algorithm 3: Progression extended for MSTL

function PROGRESS $(\phi, t, \alpha\ \Delta, \mathcal{D})$:
if $\phi = \phi_1 \wedge \phi_2$ **then**
  return $\texttt{PROGRESS}(\phi_1, t, \alpha, \Delta, \mathcal{D}) \wedge \texttt{PROGRESS}(\phi_2, t, \alpha, \Delta, \mathcal{D})$
else if $\phi = \neg\phi_1$ **then**
  return $\neg\texttt{PROGRESS}(\phi_1, t, \alpha, \Delta, \mathcal{D})$
else if $\phi = \phi_1\ \mathcal{U}_I\ \phi_2$ **then**
  if $I < 0$ **then**
    return False
  else if $0 \in I$ **then**
    return
     $\texttt{PROGRESS}(\phi_2, t, \alpha, \Delta, \mathcal{D}) \vee (\texttt{PROGRESS}(\phi_1, t, \alpha, \Delta, \mathcal{D}) \wedge \phi_1\ \mathcal{U}_{I-\Delta}\ \phi_2)$
  else
    return $\texttt{PROGRESS}(\phi_1, t, \alpha, \Delta, \mathcal{D}) \wedge \phi_1\ \mathcal{U}_{I-\Delta}\ \phi_2$
  end
else if $\phi = \forall x[\phi_1]$ **then**
  return $\bigwedge_{c \in \mathcal{D}} \texttt{PROGRESS}(\phi_1(x/c), t, \alpha, \Delta, \mathcal{D})$
else if $\phi = \exists x[\phi_1]$ **then**
  return $\bigvee_{c \in \mathcal{D}} \texttt{PROGRESS}(\phi_1(x/c), t, \alpha, \Delta, \mathcal{D})$
else if $\phi = \bigcirc^i \phi_1$ *where* $i > 0$ **then**
  return $\bigcirc^{i-1} \phi_1$
else if $\phi = R(\bigcirc^i x, \bigcirc^j y)$ *is a spatial relation where* $i > 0$ *or* $j > 0$ **then**
  return $R(\bigcirc^{i-1} x, \bigcirc^{j-1} y)$
else
  if $t, \alpha \models \phi$ **then**
    return True
  else if $t, \alpha \not\models \phi$ **then**
    return False
  else
    return Unknown
  end
end

available. Instead, we have to rely on partial information on spatial relations, which may not be sufficient to determine the exact spatial relation between a pair of spatial objects even after application of the algebraic closure for RCC-8. In those cases, Algorithm 3 will return Unknown. Assuming full knowledge, however, we are able to show its correctness as follows.

Theorem 6 (Correctness of PROGRESS for MSTL)**.** *Assuming full knowledge of the topological space through* α*, the* PROGRESS *algorithm for* MSTL *(Algorithm 3) is* correct*, meaning*

$$\mathcal{M}, t \models \phi \text{ iff } \mathcal{M}, t+1 \models \textit{PROGRESS}(\phi, t, \Delta, \mathcal{D}) \tag{3.40}$$

for wff ϕ*, time-point* t*, duration* Δ*, and domain of discourse* $\mathcal{D}$*.*

Proof. We base this proof on the correctness proof of PROGRESS on MTL, which was based on the semantics of the logic. Since MTL was extended to MSTL by the addition of spatial relations, the cases covered by lines 18–29 need to be proven.

Consider first the base case from lines 22–29. Assume first that $\mathcal{M}, t \models \phi$. Then PROGRESS($\phi$, t, α, Δ, $\mathcal{D}$) returns True on line 24, and

$$\mathcal{M}, t+1 \models \texttt{PROGRESS}(\phi, t, \alpha, \Delta, \mathcal{D}) \tag{3.41}$$

holds. Conversely, assume the latter holds. This can only be the case if it has returned True, which it only does if $t, \alpha \models \phi$, so $\mathcal{M}, t \models \phi$ holds. Thus the correctness equation holds for the base case.

Now consider the case of $\phi = R(\bigcirc^i x, \bigcirc^j, y)$ where $i > 0$ or $j > 0$. Assume first that $\mathcal{M}, t \models R(\bigcirc^i x, \bigcirc^j, y)$. Then PROGRESS returns $R(\bigcirc^{i-1} x, \bigcirc^{j-1}, y)$. From the semantics of the 'next' operator we know that our assumption yields $R(\bigcirc^{i-1} x, \bigcirc^{j-1}, y)$ for the next time-point;

$$\mathcal{M}, t+1 \models R(\bigcirc^{i-1} x, \bigcirc^{j-1}, y). \tag{3.42}$$

This corresponds to the result obtained from PROGRESS;

$$\mathcal{M}, t+1 \models \texttt{PROGRESS}(R(\bigcirc^i x, \bigcirc^j, y), t, \alpha, \Delta, \mathcal{D}). \tag{3.43}$$

Conversely, starting with the aforementioned equation, we again obtain $R(\bigcirc^{i-1} x, \bigcirc^{j-1}, y)$ for time-point $t+1$, which from the semantics of the 'next' operator leads back to $\mathcal{M}, t \models R(\bigcirc^i x, \bigcirc^j, y)$. Thus the correctness equation holds for the case of $\phi = R(\bigcirc^i x, \bigcirc^j, y)$ where $i > 0$ or $j > 0$.

Now consider the case of $\phi = \bigcirc^i \phi_1$ where $i > 0$. Assume first that $\mathcal{M}, t \models \bigcirc^i \phi_1$. Then PROGRESS returns $\bigcirc^{i-1} \phi_1$. From the semantics of the 'next' operator we know that our assumption yields $\bigcirc^{i-1} \phi_1$ for the next time-point;

$$\mathcal{M}, t+1 \models \bigcirc^{i-1} \phi_1. \tag{3.44}$$

This corresponds to the result obtained from PROGRESS;

$$\mathcal{M}, t+1 \models \texttt{PROGRESS}(\bigcirc^i \phi_1, t, \alpha, \Delta, \mathcal{D}). \tag{3.45}$$

The converse follows the same pattern in reverse order in the same way as in the previous case. Thus the correctness equation holds for the case of $\phi = \bigcirc^i \phi_1$ where $i > 0$.

The remaining cases are covered by the correctness proof for MTL by Bacchus and Kabanza (1998). By exhaustive proof over the semantics of MSTL we therefore conclude that

$$\mathcal{M}, t \models \phi \text{ iff } \mathcal{M}, t+1 \models \texttt{PROGRESS}(\phi, t, \Delta, \mathcal{D}) \tag{3.46}$$

for wff ϕ, time-point t, duration Δ, and domain of discourse $\mathcal{D}$. This means that PROGRESS for MSTL is correct. □

3.5 Performance evaluation

Thus far we introduced a logic for spatio-temporal stream reasoning and a number of methods for the evaluation of formulas in that logic. In terms of performance, we are interested in three properties. Progression allows us to evaluate MSTL formulas, which can grow exponentially in the worst case. Therefore the computational resource usage of progression is of interest. In additional to those performance experiments, we provide experimental results supporting our effectiveness claims with regards to landmark usage and state stream augmentation.

3.5.1 CPU usage of progression

The performance of progression has been studied previously (Doherty et al., 2009), but is included for the sake of completeness using our new implementation. Note that the performance of progression itself is not directly affected by supporting spatial reasoning, as it is a syntactic approach. Instead, the generation of a suitable state stream involves spatial inference so that complete states can be generated. The performance of progression is closely tied to the formula being progressed and the stream used for the progression. The evaluation of progression is therefore done through two separate experiments with different formulas and different streams. The results are shown in Figures 3.6 and 3.7. In each case, 1000 formulas are progressed concurrently.

In the first experiment we measure the CPU usage over successive progressions for the progression of the formula

$$\Box\Diamond_{[0,1000]}p, \tag{3.47}$$

where the truth value of p is determined by a regular pattern and every time-step takes 100ms. The first pattern is illustrated by an uninterrupted (red) line; the second pattern is illustrated by a dashed (green) line. For the first pattern, p is set to always be true. This corresponds to a state stream in which for every state p is set to true. The nesting of temporal operators is important here. Since p is always true, the eventually operator immediately evaluates to true as well, so the formula ceases to grow in size. Figure 3.6 shows that progression steps take a bit over 20μs per formula. In contrast, the second pattern shows a sequence wherein p is false for 10 time-steps and becomes true for one time-step, before repeating. This means that the formula must grow in order to keep track of the eventually operator, for which the interval allows a delay of up to 1000ms before p has to be true in order for the formula to not be evaluated to false. The pattern uses the full duration allowed, and once p becomes true the formula shrinks again. This shrinking and growing behaviour can be correctly observed in Figure 3.6, where the shrinking occurs every 1000ms. In the worst case for this type of formula, a progression step takes about 45μs per formula.

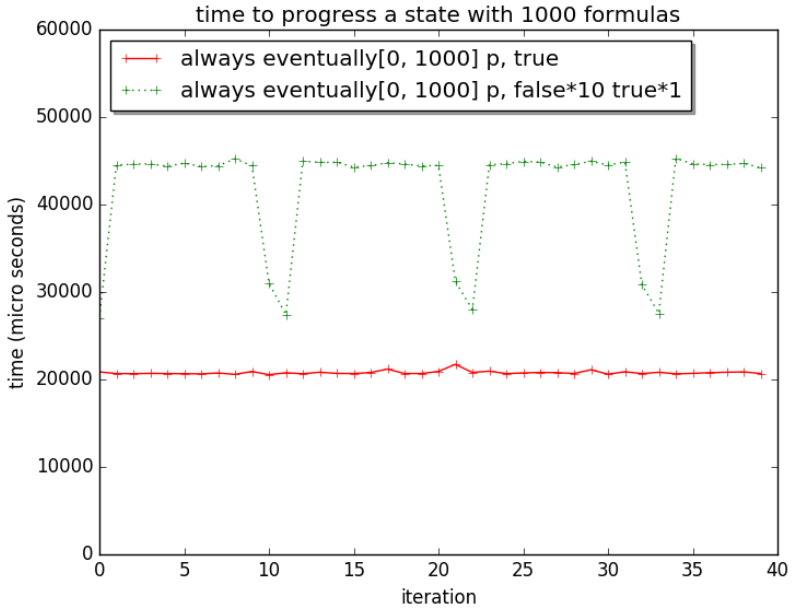

Figure 3.6: CPU usage over successive progressions when progressing $\Box\Diamond_{[0,1000]} p$ over regular state sequences.

In the second experiment we similarly measure the CPU usage over successive progressions for the progression of the formula

$$\Box\neg p \rightarrow \Diamond_{[0,1000]}\Box_{[0,999]} p, \tag{3.48}$$

where the truth value of p is again determined by a regular pattern and every time-step takes 100ms. Due to the logical nature of implication, this formula only grows whenever p is false. The different state sequences show different degrees growth accordingly. In the best case, p is never false and the formula is never expanded, resulting in progression steps taking about a constant 50μs per formula. If p does become false, the formula is expanded, and progression steps take more time. In these examples, progression steps take about 100μs per formula in the worst case.

3.5.2 Effectiveness and scalability of landmarks

In order to empirically evaluate MSTL with landmarks we ran experiments to test the effectiveness and the scalability of the landmark based approach compared to the case where no landmarks were used. In these experiments, we were only interested in consistent scenarios, to capture the operational

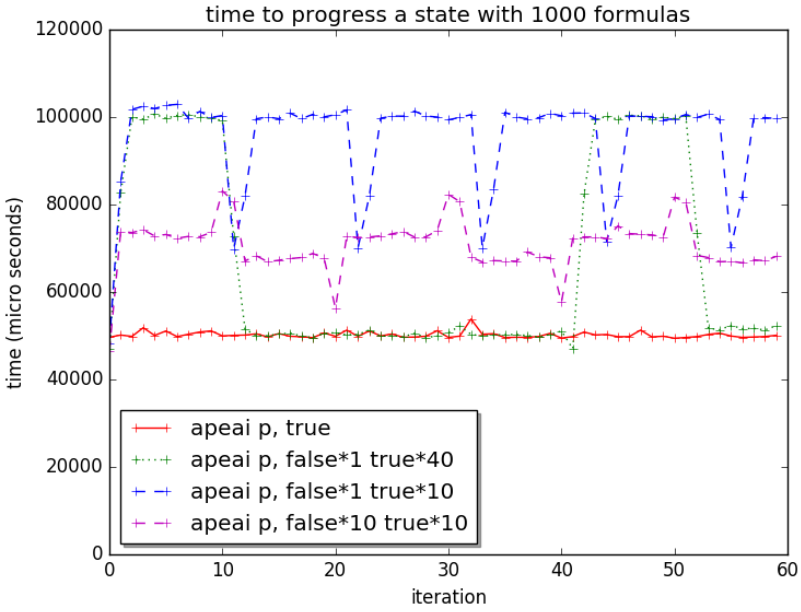

Figure 3.7: CPU usage over successive progressions when progressing $\Box\neg p \rightarrow \Diamond_{[0,1000]}\Box_{[0,999]}p$ over regular state sequences.

real-world domain. In particular, we are interested in the effects of landmarks on the resulting intertemporal disjunction size for non-landmark to non-landmark relations.

When considering two time-points t_1 and t_2, the problem of generating scenarios is given a consistent scenario with landmarks for time-point t_1 generate a consistent scenario with those same landmarks for time-point t_2. To achieve this, we make use of a variation of the scenario generation method presented by (Renz and Nebel, 2001), which was previously extended to handle static regions (Heintz and de Leng, 2014). Scenarios for a single time-point are generated based on the number of (non-landmark) regions n and the average disjunction size l. We extend this by also considering the number of landmarks m such that $n + m = |\mathcal{R}|$, and fixing parameter $l = 4$. The reason for fixing $l = 4$ is that it provides a middle ground between fully known and fully unknown. Our parameter combinations consist of varying numbers of regions between 20 and 200 with step size 20, and varying landmark ratios relative to the number of regions (i.e. m/n) between 0 and 0.9 with step size 0.1.

The initial 'seed' for a scenario covers the landmark regions and their relations to each other. In our experiments we generated 30 such seeds per parameter combination. Here we are only interested in a consistent

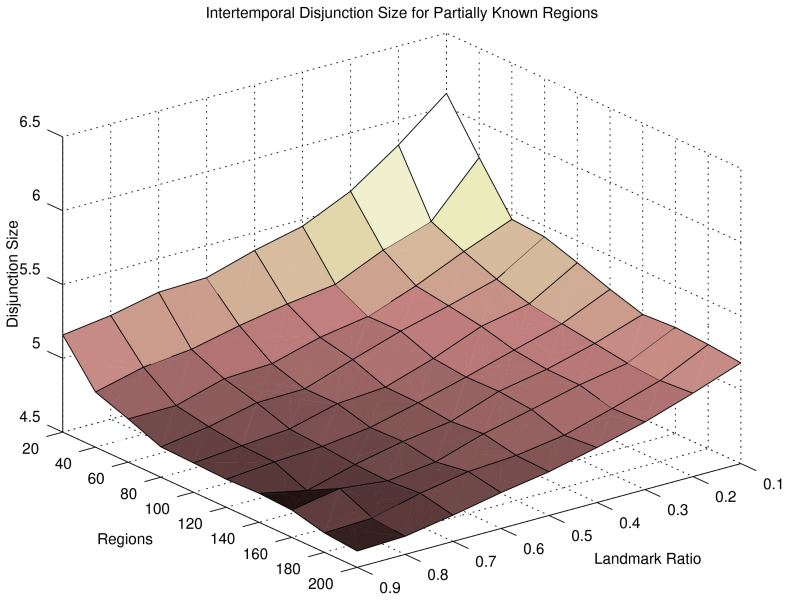

Figure 3.8: Absolute disjunction size for varying number of regions and landmark ratio; smaller is better.

scenario with complete knowledge, so GQR is used to generate consistent interpretations of scenarios. These fully known seeds can then be used as the basis for a larger spatial relation matrix by adding further regions until we obtain the desired $|\mathcal{R}|$ regions. The number of CSPs generated from a seed was kept constant at 20. Note that these CSPs then all share a seed as a common component. We can therefore combine two CSPs that share a common seed. Excluding combinations that involve the same CSP twice, given 30 seeds and 20 CSPs per seed we get $30 \times (20 \times (20 - 1))/2 = 5700$ instances for each parameter set.

The results of our experiments are illustrated in Figures 3.8 and 3.9, where every point represents the average over 5700 instances. In Figure 3.8, the number of regions and the landmark ratio are changed to see how they affect the disjunction size of non-landmark to non-landmark spatial relations. Here we limit ourselves to the average over the spatial relations that are not fully unknown. The results show that the more landmarks are added, the less uncertainty in terms of disjunction size is measured for these relations, reaching between disjunction sizes 4 and 5 for a landmark ratio of 0.9. The landmark approach is also scalable in terms of the number of regions.

This is also shown in Figure 3.9, which illustrates the percentage of

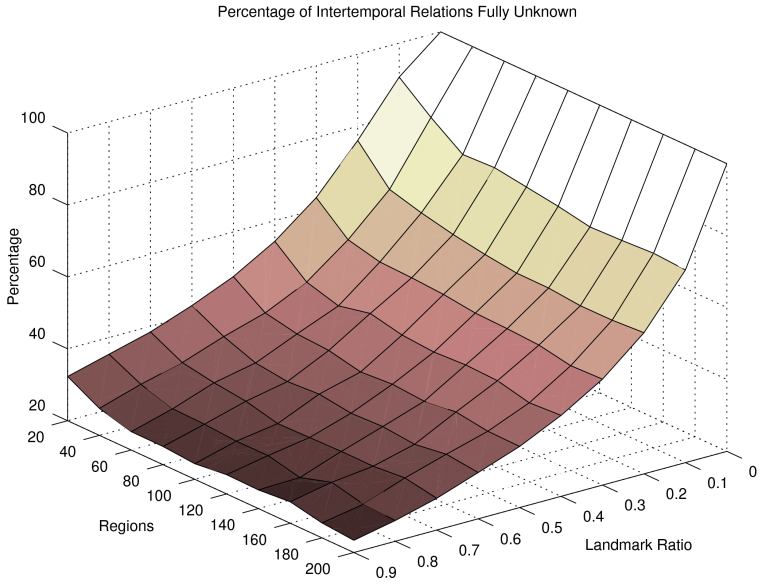

Figure 3.9: Percentage of such relations fully unknown.

non-landmark to non-landmark intertemporal relations that remain fully unknown. Previously, we could not say anything about these relations, as illustrated by the percentage of fully unknown relations being 100%. Using landmarks, this is reduced to 30% for landmark ratio 0.9, but having a landmark ratio as low as 0.1 results in an improvement of roughly 20%.

3.5.3 Caching spatial relations between rigid objects

The spatial reasoning is mainly dependent on the number of variables, the average number of constraints (degree) and the average label size (Renz and Nebel, 2001). Using basically the same method as Renz and Nebel (2001) we evaluate the effect of precomputing the algebraic closure of the static variables, compared to computing the whole algebraic closure for each time-step.

In the experiments we try to estimate the function $A'(v, E(deg), E(l), r)$ by measuring the execution time on instances with the number of variables v, the expected degree $E(deg)$, the expected label size $E(l)$ and the ratio of dynamic variables r. The number of variables can be divided in a dynamic part $v_d = r \times v$ and a static part $v_s = v - v_d$. The expected degree is the expected number of relations from a given dynamic variable to other variables. The expected label size is the expected size of the disjunction of

RCC-8 relations for a given relation between a dynamic variable and some other variable. In this evaluation we use $E(l) = 4.0$.

Because the static component only has to be computed once, we compare the case where all variables are dynamic to the case where there is a separation between static and dynamic variables, ignoring the time it takes to compute this static component. The mean performance results of the former are denoted by $A(v, E(deg), 4.0)$. For the mean performance results of the dynamic component of the latter, the notation $A'_d(v, E(deg), 4.0, r)$ is used. The performance experiments used values of $E(deg)$ ranging from 1 to 20 with step size 1, and values of v ranging from 20 to 500 with step size 20. The value of r was chosen to be constant, $r = 0.25$. For every combination, we took the population mean CPU time over 100 runs. The population mean was chosen to account for the difference in distribution between the satisfiable and unsatisfiable problem instances.

The evaluation compares the case of all variables being dynamic to the case when some are static. A selection of the evaluation results are shown in Figures 3.10 and 3.11.

The top graph in Figure 3.10 shows the absolute performance in CPU time of $A(v, E(deg), 4.0)$. The graph shows a ridge at $E(deg) = 9$. This is where the phase-transition occurs, where the majority of problem instances flip from being satisfiable to being unsatisfiable. In comparison, the bottom graph shows the absolute performance in CPU time of $A'_d(v, E(deg), 4.0, 0.25)$. Note that this only shows the time needed by the dynamic component. For low degrees, the time needed surpasses that of the exclusively dynamic case. A potential explanation for this behaviour is that the combination of a low degree and high number of variables for the dynamic variables combined with the completely known static part (i.e. a degree of $v_s - 1$ and expected label size $E(l) = 1$ for the static component) makes for computation-intensive problem instances. For all other values of v and $E(deg)$ the performance is significantly improved.

A comparison of the two top-row graphs is shown in Figure 3.11. The comparison shows a clear decrease in performance when comparing the exclusively dynamic case to the separated case when the degree is low and the number of variables is high. However, in all other cases there is a performance increase, especially around the phase-transition area. The general performance increase is roughly 50 milliseconds. The relative performance increase shows an increase of about 35% in the phase-transition area, and an increase of close to 100% for a low number of variables.

The results show that the separation of dynamic and static variables for $r = 0.25$ generally leads to better performance, except in the case of a low degree with a high number of variables. The performance increase is at its highest around the phase-transition region where the more difficult problem instances reside. The performance increase is expected to be higher for lower values of r and decrease as r approaches 1.

3.6 Open problems

- While the 'next' operator adds powerful additional expressivity in terms of intertemporal spatial relations, its semantics is fixed to temporal states, which in practice means that 'next' refers to different lengths of time for different stream frequencies. A formula therefore has a different meaning depending on what type of stream is used in terms of frequency. Can this ambiguity be resolved in a meaningful way?
- The semantics of MSTL makes use of a topological space and a mapping α from region symbols and time-points to this topological space. When evaluating MSTL formulas, however, this mapping cannot be used directly, because it would require some oracle. Instead, we make use of spatial relations that are part of a state stream. How can the semantics of MSTL be modified to use these relation-based spatial states rather than a topological space?
- Landmarks have been shown to reduce the uncertainty of intertemporal spatial relations. However, in many cases this means that formulas referring to intertemporal relations will still be evaluated to unknown. Can the temporal modality be complemented with the classical necessity modality to reason about possible consistent interpretations instead?
- Can the qualitative representational approach for MSTL be generalised to types of qualitative relations other than spatial relations?

3.7 Summary

While spatial extension to temporal reasoning have been investigated in the past, these works have not specifically focused on the application of these resulting spatio-temporal logics in a stream reasoning context. We presented MSTL, a metric spatio-temporal logic that combines the well-known MTL temporal logic and RCC-8 qualitative spatial calculus. Similar to ST_1, MSTL allows for the application of the 'next' operator to region terms, which makes it possible to express intertemporal spatial relations between regions. Since qualitative intertemporal spatial relations cannot be observed directly, a frame of reference formed by landmark regions is used to reduce the uncertainty of the intertemporal spatial relations. To facilitate incremental reasoning over streams, the generation of state streams is discussed. These state streams are used to progress MSTL formulas using an extension of the classical progression procedure for MTL. This makes it possible to apply model checking to MSTL formulas, which is useful in applications such as execution monitoring.

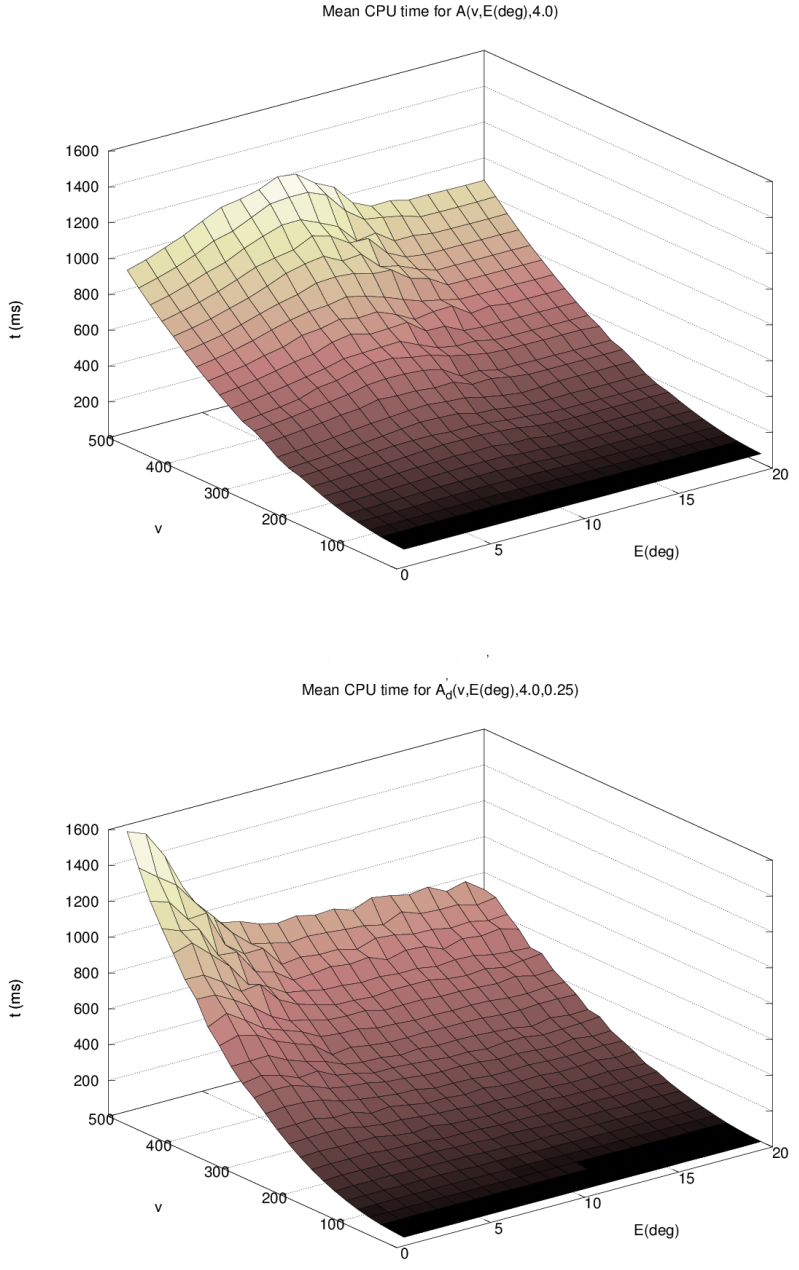

Figure 3.10: Comparison of mean CPU times when separating static and dynamic variables.

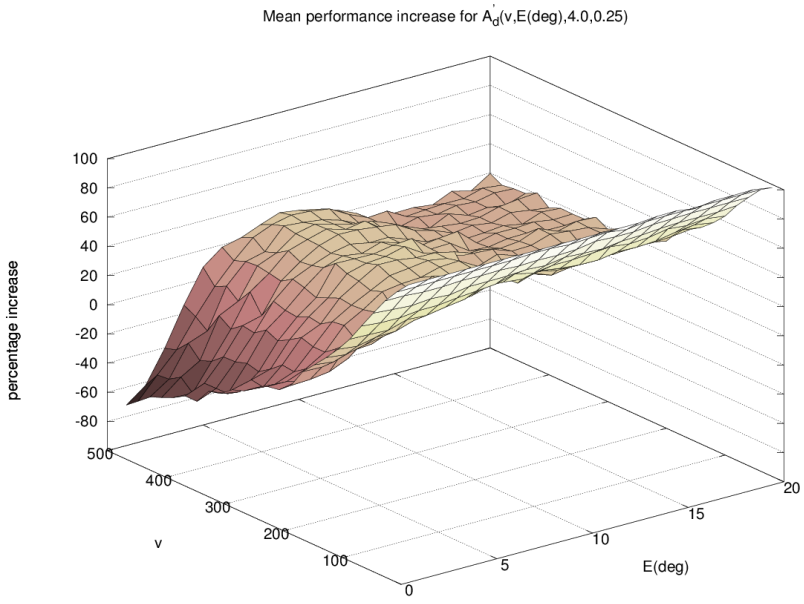

Figure 3.11: Relative CPU time increase when separating static and dynamic variables.

Chapter 4

Semantic subscriptions

Logic-based stream reasoning commonly makes use of temporal logics to express statements concerning the truth value of properties over time. Stream reasoning techniques usually do not consider where their data originates from, and assume it to be given. However, the generation of streaming data for the purpose of stream reasoning is an important stream processing task. State stream generation is the refining of low-level data streams into high-level symbolic information that can be combined for consumption by such a stream reasoning tool. This chapter borrows from and extends previous work on state stream generation (de Leng, 2013; Heintz and de Leng, 2013; de Leng and Heintz, 2014), configuration modelling and planning (de Leng and Heintz, 2015b,a, 2017), and unpublished improvements for the reconfiguration procedure.

4.1 Introduction

Robotic systems are getting increasingly complex, with more and more components usually connected by some form of publish-subscribe messaging pattern. Support for this type of integration is often provided by middleware such as CORBA and the Robot Operating System (ROS). The configuration of what channels a component publishes and subscribes to is often done manually or through scripts. This is both error-prone and assumes that the set of available components does not change at run-time. However, IoT development towards for example swarmlets (Latronico et al., 2015) points to a future in which systems are increasingly heterogeneous, decentralised and geographically spread-out. The assumption of an unchanging or slowly changing set of available components is therefore rapidly becoming unreasonable.

The challenge of dealing with this volatility also affects the task of generating state streams for the purpose of progression. After all, if component sets cannot be assumed to be constant, the task of generating a state

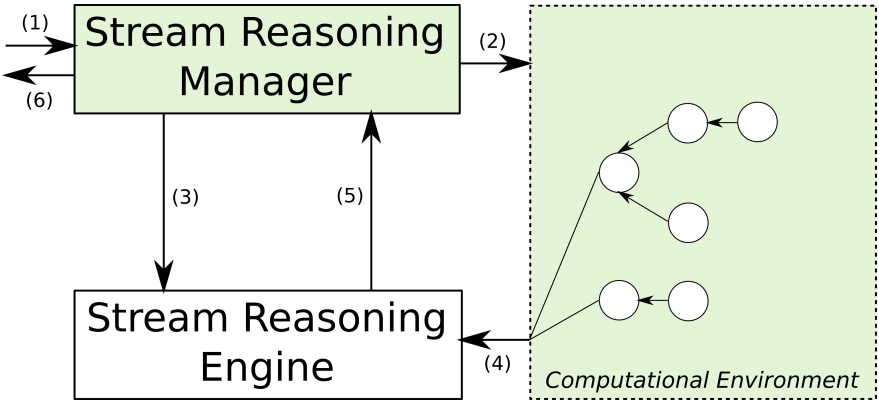

Figure 4.1: Conceptual overview with the adaptive state stream generation highlighted.

stream needs to be complemented with the task of maintaining one. The second strand of this thesis focuses on exactly this problem. The conceptual overview from before, with the relevant state stream generation components highlighted, is shown in Figure 4.1. It illustrates how the stream reasoning manager accepts a spatio-temporal logic formula (1) and reconfigures the computational environment (2) in order to generate a state stream (4) for the stream reasoning engine so it can evaluate that formula. Once the configuration is initiated, the stream reasoning manager keeps track of the computational environment and reconfigures it when needed. To see how this problem relates to reasoning over streams, consider the previous chapter on a logic for spatio-temporal stream reasoning. The task of the stream reasoning engine is to evaluate a formula. In order to do so, the symbols in the formula need to be interpreted. This is done by grounding these symbols into streaming data, which is produced by an underlying computational environment. This thesis explicitly assumes that the environment may change over time, which makes maintaining a stream for formula evaluation challenging.

In this chapter, the problem of adaptively generating a state stream is translated into the problem of robustly satisfying a semantic subscription in a stream reasoning framework. We first consider a formalisation of a stream reasoning framework and its dynamics, called the *DyKnow model*, which allows us to frame the problem as an optimisation problem. The purpose of the formal model is to be general enough such that implementation details are abstracted away, allowing for potentially many different realisations. With the DyKnow model formalised we consider a common representation of configurations relative to an ontology. We then consider a life-cycle and algorithms for setting up and maintaining semantic subscriptions, finalising the formalisation of semantic subscriptions.

4.1.1 Semantic integration

The integration of different components based on their semantics is referred to as *semantic integration* and is closely related to the work presented here. An approach to 'semantically-enabled sensor plug & play' was proposed by Bröring et al. (2009), who identified challenges to achieving sensor plug-and-play based on semantic knowledge of sensor observations. They subsequently proposed a method for automatic plug-and-play functionality by making use of a Sensor Bus (Bröring et al., 2011) that matches services to sensors. The approach to semantic subscriptions taken in this thesis is more advanced than the Sensor Bus approach in that we periodically recombine and reconnect components whereas the Sensor Bus directly connects with information sources.

Another example is research towards Semantic Sensor Networks, which led to the development of the Semantic Sensor Network ontology (SSN) (Compton et al., 2012). SSN focuses on well-structured semantic descriptions of sensors. The work presented here makes use of semantic descriptions of streaming components rather than sensors by using functional descriptions of the inputs and outputs of these components. These functional descriptions are extensions of the OWL-S service ontology (Martin et al., 2004) applied to a streaming context.

4.1.2 Configuration planning

The aforementioned reconfiguration capabilities are also closely related to configuration planning. Automatic (re)configuration techniques have been studied in detail (Rao and Su, 2005; Dustdar and Schreiner, 2005; Pejman et al., 2012). The work by Tang and Parker (2005) on ASyMTRe is an example of a system geared towards the automatic self-configuration of robot resources in order to execute a certain task. Similar work was performed by Lundh et al. (2008) related to the Ecology of Physically Embedded Intelligent Systems, also called the PEIS-ecology (Saffiotti et al., 2008). Given a high-level goal describing a task, Lundh et al. use a configuration planner to configure a collection of robots towards the execution of the task rather than logic-based stream reasoning. Their solution is however designed for use within the PEIS middleware and does not easily transfer to other environments such as the ROS middleware. Lundh (2009) further points out that their approach uses static cost measures and could benefit from incorporating semantic knowledge. Our approach focuses on a more advanced representation of cost, and makes use of semantic descriptions for components.

The SAMSON Wireless Sensor Networks (WSNs) middleware by Portocarrero et al. (2016) is similar to run-time reconfigurable systems in considering a dynamic environment in which a network can be reconfigured to deal with changes, albeit at a lower level. In the case of SAMSON, these changes include faults, but also disconnection and power concerns. A sur-

Symbol	Description
$l_i \in \mathsf{Var}$	Set of variables
$tag, itag_i, otag \in \mathsf{Tag}$	Set of tags
$v_i \in \mathcal{V}$	Set of (structured) values
$t_i \in \mathcal{T}$	Set of time-points
$tid, cid, qid \in \mathbb{N}$	Set of identifiers
$in_i, out, chan \in \mathbb{N}$	Set of channels
$\left\langle cid, tid, [in_1, in_2, \ldots, in_n]^T, out, \mathcal{S} \right\rangle \in CU$	Computation units
$\left\langle tid, f(x_1, \ldots, x_n, \mathcal{S}), [itag_1, \ldots, itag_n]^T, otag \right\rangle \in F$	Transformations
$\langle qid, tag, chan \rangle \in T$	Targets
$\mathcal{S} \subseteq \mathsf{Var} \times \mathcal{V}$	States
$\sim \subseteq \mathsf{Tag} \times \mathsf{Tag}$	Similarity relation
$f : \mathcal{V}^n \times \mathcal{S} \hookrightarrow \mathcal{V} \times \mathcal{S}$	Transformation function
$\varepsilon = \langle CU, F, T, \sim \rangle$	Environment
$\delta = (CU^+, CU^-, F^+, F^-, T^+, T^-)$	Change set
$\varepsilon' = \varepsilon \otimes \delta$	Update
$\varepsilon \Rightarrow_\delta \varepsilon'$	
$\varepsilon \in \mathsf{Valid}$	Set of valid environments

Table 4.1: Notation for the DyKnow model.

vey of other recent work towards WSN middlewares is presented by Kerasiotis et al. (2015).

4.2 DyKnow model

The *DyKnow model* is a formalisation of stream reasoning frameworks and extends earlier work by Heintz (2009) and Heintz et al. (2010) which considered such frameworks to be composed of possibly many interconnected components. The formal model is general and serves as a specification from which potentially many different realisations can be created. Table 4.1 provides a complete summary for the notation used in describing the model.

4.2.1 Streams

State streams as expected by progression contain the state information of fluents for all time-points under consideration. A state stream is a specialisation of a *stream*, which we consider to be a sequence of time-stamped values.

Definition 16 (Stream)**.** *A* stream *is an unbounded sequence of time-stamped values*

$$((l_0, v_0, t_0), (l_1, v_1, t_1), \ldots) \tag{4.1}$$

where $v_i \in \mathcal{V}$ represents a (structured) value, $l_i \in \mathsf{Var}$ represents a variable name, and $t_i \in \mathcal{T}$ represents a time-point.

The individual triplets that make up a stream are referred to as *samples*. In the DyKnow model, streams are closely tied to *channels*, which provide a transportation mechanism for streams to 'flow' over. Channels are used to connect components that produce and consume samples over time, effectively consuming and producing streams. The DyKnow model makes use of a computation-centric view, wherein streams are implicit products of explicit configurations of computations.

4.2.2 Computational environment

A *computational environment* is composed of a *computation graph*, *transformations* and *targets*. The computation graph consists of *computation units* connected by *channels*.

Streams are the product of transformations, which can either refine existing streams into new streams, or act as sources by generating streams without requiring any input streams. In practice, sources often use information external to the computational environment to generate streams, for example through sensor observations. A transformation is considered to be an annotated function that can be instantiated as a computation unit for application within a specific configuration.

Definition 17 (Transformation)**.** *A* transformation *(TF) is an annotated stream-generating function that takes streams as inputs. It is described by a tuple*

$$\left\langle tid, f(x_1, \ldots, x_n, \mathcal{S}), [itag_1, \ldots, itag_n]^T, otag \right\rangle, \tag{4.2}$$

where $tid \in \mathbb{N}$ *represents a unique transformation identifier,* $f : \mathcal{V}^n \times \mathcal{S} \hookrightarrow \mathcal{V} \times \mathcal{S}$ *represents a partial function from input values and an initial state to an output value and a resulting state,* $itag_i \in \mathsf{Tag}$ *represent tags for inputs, and* $otag \in \mathsf{Tag}$ *represents the output tag.*

Definition 18 (Computation unit)**.** *A* computation unit *(CU) is a component that is described by a tuple*

$$\left\langle cid, tid, [in_1, in_2, \ldots, in_n]^T, out, \mathcal{S} \right\rangle, \tag{4.3}$$

where $cid \in \mathbb{N}$ *represents a unique identifier for CUs,* $tid \in \mathbb{N}$ *represents the unique identifier of the transformation which this CU is an instance of,* $in_i \in \mathbb{N} \cup \{\mathsf{none}\}$ *represent incoming channels,* $out \in \mathbb{N} \cup \{\mathsf{none}\}$ *represents the outgoing channel, and* $\mathcal{S} \subseteq \mathsf{Var} \times \mathcal{V}$ *represents the state as a relation between variables and values.*

Note that there is a close relation between CUs and TFs—a CU is called an *instance* of a TF iff its *tid* identifiers match.

Example 10 (TFs and CUs)**.** *Robots commonly use visual sensing methods to detect and track objects of interest. Consider a* ball detector *that is able to detect footballs by their round white shape with black spots. The ball detector can*

be represented in terms of a transformation *and a* computation unit. *The ball detector transformation refers to the mathematical function describing the detection method, together with meta-information for this function. It is annotated with tags describing its input as camera images, and its output as bounding boxes. We can apply the transformation by connecting it to an input stream of camera images, yielding a stream of bounding boxes. This application of the transformation is called a computation unit. Every CU has an identity, a reference to its corresponding TF, connections to input and outputs channels, and state information. The state information allows the transformations to be stateful, meaning they can retain information that makes it easier to for example perform tracking after an initial detection.*

Lastly, the computational environment contains *targets*, which describe semantic subscriptions for outside modules such as the stream reasoning engine. Note that subscriptions also occur *within* the computational environment, but that these are not referred to as targets because they do not reflect the global configuration goals of the computational environment. Subscriptions of the latter kind are described by the connections between CUs and channels as shown earlier.

Definition 19 (Target)**.** *A* target *describes a desired semantic subscription and is denoted by a tuple*

$$\langle qid, tag, chan \rangle \,, \tag{4.4}$$

where $qid \in \mathbb{N}$ *is a unique (query) identifier,* $tag \in \mathsf{Tag}$ *is a description of the desired information, and chan is the channel the described stream is expected on.*

Targets thus indirectly represent configuration goals for the computational environment[6] by indirectly referencing desired streams by their semantic descriptions. These streams are generated by instantiated transformations, which in turn have input requirements. For a given set of targets, there may be many different computation graphs which satisfy all of the input requirements and similarity relations at different costs.

Example 11 (Targets)**.** *Consider the following statement: "Whenever a NAO robot has its battery level drop under 10%, it should be in the charging state within the next minute." Such a statement could be logically formalised as*

$$\forall x \in \mathsf{NAO} \left[\Box \left(\mathsf{batteryLevel}(x) < 0.1 \rightarrow \Diamond_{[0,60]} \mathsf{Charging}(x) \right) \right]. \tag{4.5}$$

In order to evaluate this formula, we need battery level information and charging state information for NAO robots. Targets *allow us to specify what kind of information we are interested in with the help of tags. To evaluate this formula, we thus need two targets; one pertaining to battery level information and another to charging state information. The symbols in the formula can then be interpreted by grounding them to the streaming data, utilising the channel information which is part of every target.*

[6] Alternatively, one can consider targets to represent constraints on channels. These constraints are then described in terms of desired semantic descriptions.

By combining these elements, we can formally describe the computational environment by an *environment*.

Definition 20 (Environment)**.** *An* environment *is denoted by a tuple*

$$\varepsilon = \langle CU, F, T, \sim \rangle , \tag{4.6}$$

where CU denotes a set of computation units called a computation graph*, F denotes a collection of transformations called a* library*, T denotes a set of targets called a* goal*, and $\sim \subseteq \mathsf{Tag} \times \mathsf{Tag}$ denotes a* similarity relation *between tags. Elements of environment ε have short-hand representations CU_ε, F_ε, T_ε, and $\sim_\varepsilon$ respectively.*

An environment thus encodes the configuration of the system as well as the state of its individual components. It is connected to streams through the collection of channels that connect the various CUs, because they are a product of those CUs. There is therefore a total mapping from streams to channels. Since CUs define outgoing and incoming channels, there is a clear connection between streams and their source and destination CUs as well.

4.2.3 Dynamics

An environment is a representation of the state of the configuration of the computational environment. This environment may be subjected to changes over time. These changes are represented by a *change set*.

Definition 21 (Change set)**.** *A* change set *is a tuple*

$$\delta = (CU^+, CU^-, F^+, F^-, T^+, T^-) \tag{4.7}$$

consisting of set additions and set removals denoted by superscript '$+$' and '$-$' respectively. The notation $\delta_\varnothing$ is used as a short-hand to describe the absence of change, i.e. $\delta_\varnothing = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$.

Change sets can thus add and remove elements to and from the environment. These additions and removals can also be used to for example represent tag changes in transformations or connection changes of CUs to channels. Whenever an environment changes in a way that can be represented using a change set, we call this change an *update*. More formally, an update is the application of a change set to an environment, yielding a new environment.

Definition 22 (Update)**.** *An* update *applying a change set δ to an environment ε is denoted by $\varepsilon' = \varepsilon \otimes \delta$ (alternatively: $\varepsilon \Rightarrow_\delta \varepsilon'$), where $\otimes$ maps environments ε and change set δ to resulting environments ε' such that*

$$CU_{\varepsilon'} = (CU_\varepsilon \cup CU_\delta^+) \setminus CU_\delta^- , \tag{4.8}$$

$$F_{\varepsilon'} = (F_\varepsilon \cup F_\delta^+) \setminus F_\delta^- , \tag{4.9}$$

$$T_{\varepsilon'} = (T_\varepsilon \cup T_\delta^+) \setminus T_\delta^- . \tag{4.10}$$

Change sets can be used to express operations of interest on environments. We call these operations *actions*. In particular, we are interested in the addition and removal actions for environment elements, as well as actions for changing connections between CUs and channels.

TFs are identified by a unique *tid* and describe a function $f(x_1, \ldots, x_n, \mathcal{S})$ from inputs and current state to an output and resulting state. They are further annotated with tags in Tag for the inputs and the output. Common actions affecting TFs in a computational environment are *register* and *deregister*.

Definition 23 (Register action)**.** *The* register action *covers the class of change sets defined by the function*

$$register(\varepsilon, tid, f, \boldsymbol{itag}, otag) = (\varnothing, \varnothing, \{\langle tid, f, \boldsymbol{itag}, otag\rangle\}, \varnothing, \varnothing, \varnothing). \quad (4.11)$$

Definition 24 (Deregister action)**.** *The* deregister action *covers the class of change sets defined by the function*

$$deregister(\varepsilon, tid) = (\varnothing, \varnothing, \varnothing, F, \varnothing, \varnothing), \quad (4.12)$$

where $F = \{\langle tid, \_, \_, \_\rangle \in F_\varepsilon\}$ *and* $\_$ *represents a wildcard.*

Targets are composed of a (query) identifier, tag, similarity relation, and a specified channel. Like TFs, targets can be added and removed by the *query* and *release* actions.

Definition 25 (Query action)**.** *The* query action *covers the class of change sets defined by the function*

$$query(\varepsilon, qid, tag, chan) = (\varnothing, \varnothing, \varnothing, \varnothing, \{\langle qid, tag, chan\rangle\}, \varnothing). \quad (4.13)$$

Definition 26 (Release action)**.** *The* release action *covers the class of change sets defined by the function*

$$release(\varepsilon, qid) = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, T), \quad (4.14)$$

where $T = \{\langle qid, \_, \_\rangle \in T_\varepsilon\}$ *and* $\_$ *represents a wildcard.*

Like TFs and targets, CUs can also be added and removed. However, unlike with TFs and targets, existing CUs can be connected to and disconnected from channels as well. We therefore consider the addition and removal of CUs to be two actions in addition to the connecting and disconnecting of existing CUs. Adding and removing CUs is represented by the *spawn* and *destroy* actions.

Definition 27 (Spawn action)**.** *The* spawn action *covers the class of change sets defined by the function*

$$spawn(\varepsilon, cid, tid, S) = (CU, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing), \quad (4.15)$$

where $CU = \left\{\left\langle cid, tid, [\mathsf{none}, \ldots, \mathsf{none}]^T, \mathsf{none}, \mathcal{S}\right\rangle\right\}$.

Definition 28 (Destroy action)**.** *The* destroy action *covers the class of change sets defined by the function*

$$destroy(\varepsilon, cid) = (\varnothing, CU, \varnothing, \varnothing, \varnothing, \varnothing), \tag{4.16}$$

where $CU = \{\langle cid, \_, \_, \_, \_\rangle \in CU_\varepsilon\}$ *and* $\_$ *represents a wildcard.*

The spawn action thus adds a CU with a provided state to account for e.g. parameters. Since CUs encode their own connections to channels, the removal of a CU implicitly breaks any connections to channels. When the spawn action is applied, a CU is added such that all of its connections are set to none by default. This initial state can then be altered by using the *connect* and *disconnect* actions, for each of which we have to consider two variants to distinguish between inputs and output.

Definition 29 (Connect action)**.** *The* connect action *covers the class of change sets defined by the functions*

$$connect_{\downarrow}(\varepsilon, cid, i, chan) = (CU^+, CU^-, \varnothing, \varnothing, \varnothing, \varnothing), \tag{4.17}$$

where CU^+ *and* CU^- *are defined for every* $\langle cid, tid', \boldsymbol{in}', out', \mathcal{S}'\rangle \in CU_\varepsilon$ *as*

$$CU^+ = \left\{ \left\langle cid, tid', \begin{bmatrix} \vdots \\ in'_{i-1} \\ chan \\ in'_{i+1} \\ \vdots \end{bmatrix}, out', \mathcal{S}' \right\rangle \right\}, \tag{4.18}$$

$$CU^- = \left\{ \langle cid, tid', \boldsymbol{in}', out', \mathcal{S}'\rangle \in CU_\varepsilon \right\}, \tag{4.19}$$

and its outgoing variant

$$connect_{\uparrow}(\varepsilon, cid, chan) = (CU^+, CU^-, \varnothing, \varnothing, \varnothing, \varnothing), \tag{4.20}$$

where CU^+ *and* CU^- *are defined for every* $\langle cid, tid', \boldsymbol{in}', out', \mathcal{S}'\rangle \in CU_\varepsilon$ *as*

$$CU^+ = \left\{ \langle cid, tid', \boldsymbol{in}', chan, \mathcal{S}'\rangle \right\}, \tag{4.21}$$

$$CU^- = \left\{ \langle cid, tid', \boldsymbol{in}', out', \mathcal{S}'\rangle \in CU_\varepsilon \right\}. \tag{4.22}$$

Definition 30 (Disconnect action)**.** *The* disconnect action *covers the class of change sets defined by the functions*

$$disconnect_{\downarrow}(\varepsilon, cid, i) = connect_{\downarrow}(\varepsilon, cid, i, \mathsf{none}), \tag{4.23}$$

$$disconnect_{\uparrow}(\varepsilon, cid) = connect_{\uparrow}(\varepsilon, cid, \mathsf{none}). \tag{4.24}$$

Actions are useful to concisely describe common change sets, and will be used later as part of a reconfiguration algorithm.

4.2.4 Cost and optimality

While there may be many different environments that would satisfy a target, not all such environments are equally preferred. This is due to the costs associated with the run-time expenses of maintaining such a resulting environment, and the one-time expense of applying the change set that yields such a resulting environment. We refer to the cost of maintaining a CU as *upkeep*. Likewise, the cost of instantiating a CU is called *labour*. While labour is a one-time cost, upkeep accumulates over time.

The measured labour and upkeep are represented by functions from environments or change sets to cost. These global cost measures are obtained from the individual CUs.

Definition 31 (Labour)**.** Labour *is the observed non-negative cost of performing an update $\varepsilon \otimes \delta$ and is equal to*

$$labour(\delta) = \sum_{cu \in (CU^{+} - CU^{-})} labour(tid(cu)). \tag{4.25}$$

Definition 32 (Upkeep)**.** *The run-time cost of an environment $\varepsilon = \langle CU, F, T, \sim \rangle$ is referred to as* upkeep*. Upkeep represents the observed non-negative run-time cost for one time-unit and is calculated as*

$$upkeep(\varepsilon) = \sum_{cu \in CU_{\varepsilon}} upkeep(cid(cu)). \tag{4.26}$$

Labour and upkeep can be used to represent the cost of change sets and environments. This is useful when we wish to compare the costs of different (alternative) updates. We will make use of estimators $\widehat{labour}$ and $\widehat{upkeep}$ to represent the estimated rather than measured labour and upkeep of change sets and environments.

A computational environment may become invalid or suboptimal as the result of updates. This may for example happen due to changing operational costs associated with CUs (upkeep), CUs may crash and require replacing, transformations may become unavailable rendering their CU instances invalid, or new transformations may become available for a lower cost. In order to maintain adaptive semantic subscriptions, the problem is to find a change set such that, when applied to an environment, the resulting environment is valid and update is optimal.

Definition 33 (Validity)**.** *An environment ε is* valid*, denoted by $\varepsilon \in$ Valid, iff for every CU:*

1. *there exists an associated TF in F_{ε};*
2. *for every identifier in_i there exists a CU in CU_{ε} for every $1 \leq i \leq n$, i.e. no subscriptions to* none*;*

3. *for every target* $\langle qid, tag, chan \rangle$ *in* T_ε*, there exists a CU with an associated TF such that* $tag \sim_\varepsilon otag$*; and*

4. $itag_i \sim_\varepsilon otag$ *holds for every connected pair of CUs.*

We exclude change sets that yield an invalid environment when used in an update. This reduces the number of applicable change sets to just those that yield environments that satisfy all targets. A pragmatic relaxation is to also allow for change sets that satisfy some targets, if it is not possible to satisfy all targets.

By combining validity with the estimators for labour and upkeep, we obtain a cost estimator that takes into account whether the resulting environment is valid. A value `MAX_COST` is used to represent an upper limit on the cost of an update. For updates yielding invalid environments, this is represented by a cost exceeding `MAX_COST`.

Definition 34 (Cost)**.** *The cost estimator* $\widehat{cost}$ *combining estimators* $\widehat{upkeep}$ *and* $\widehat{labour}$ *is defined as*

$$\widehat{cost}(\varepsilon, \delta, H) = \begin{cases} \widehat{labour}(\delta) + H \times \widehat{upkeep}(\varepsilon \otimes \delta), & \text{if } \varepsilon \otimes \delta \in \mathsf{Valid}, \\ \texttt{MAX\_COST} + 1, & \text{otherwise.} \end{cases} \tag{4.27}$$

The cost estimator is used for determining the estimated cost of updates. An *optimal* update is one that minimises the estimated cost of applying a change set and the estimated upkeep over a predetermined horizon. It makes use of the cost estimator and excludes updates that exceed the maximum cost, for example due to being absent from Valid.

Definition 35 (Optimality)**.** *An update* $\varepsilon' = \varepsilon \otimes \delta^*$ *is* optimal *relative to a horizon of* H *time-units iff* $\delta^* \in \Delta^*$*, where*

$$\Delta^* = \arg\min_{\delta} \widehat{cost}(\varepsilon, \delta, H) \tag{4.28}$$
$$\text{subject to } \widehat{cost}(\varepsilon, \delta, H) \leq \texttt{MAX\_COST}$$

for cost estimator $\widehat{cost}$ *and upper bound* `MAX_COST`*.*

Note that there may be many optimal change sets, in which case any can be chosen. Alternatively, if no change set can make the resulting environment valid, there are no optimal change sets. The choice of horizon determines how conservative change sets are; if the horizon is large, upkeep starts to outweigh labour more than in cases where the horizon is kept short. Different estimators can be used, ranging from simplistic constant values to advanced predictive models whose accuracy is used to increase or decrease the length of the next horizon.

4.3 Handling perturbations

During the run-time of the system, it is possible for the environment to change outside of its own control. We call these changes *perturbations*, which can be represented in terms of change sets. Some perturbations can be relatively harmless; for example, a transformation that is currently not in use could be deregistered. Worse would be the case wherein a transformation for which CUs exist is deregistered. In such a case, the behaviour of those CUs becomes undefined, and they therefore require removal. Furthermore, the loss of these CUs can leave holes in the computation graph, leaving the environment invalid. In yet another example, a CU could crash and thereby be removed from the computation graph, resulting in similar potential problems. These last examples are clear cases wherein a perturbation results in an expensive and suboptimal environment. Less clear cases are those wherein new transformations become available. A new transformation could be cheaper to use than the transformations currently in use by an environment, but making this change is not critical.

Definition 36 (Perturbation)**.** *We can consider different types of* perturbations δ_p. Short-term negative perturbations[7] *result in an immediate cost increase (compared to no change) when considering an equal horizon H:*

$$cost(\varepsilon, \delta_p, H) > cost(\varepsilon, \delta_\varnothing, H). \tag{4.29}$$

When the cost does not change as the result of δ_p, it is considered to be a short-term neutral perturbation*. Similarly,* long-term positive perturbations *make possible an update that would result in a cost decrease, i.e.*

$$\exists \delta^* [cost(\varepsilon \otimes \delta_p, \delta_\varnothing, H) > cost(\varepsilon \otimes \delta_p, \delta^*, H)], \tag{4.30}$$

with (inversely) long-term neutral perturbations *lacking such an update. Different perturbations can thus have different effects in the short and long term.*

To handle both the short and long term repercussions of perturbations, semantic subscriptions are periodically evaluated and updated to repair or improve the underlying environment. This recurring process is referred to as the *configuration life-cycle*. The life-cycle is composed of a number of phases which are repeated every cycle, which starts with a *review interval* followed by a *stable interval*.

Review interval. The purpose of the review interval is to reflect on the preceding stable interval (if any) and to improve the environment configuration. During this interval, the stream reasoning manager searches for a change set such that its application to the current environment constitutes an optimal update. Whether an update is optimal is determined by

[7] *Short-term positive perturbations* are generally ignored as they would require an outside force to for example remove a target together with any CUs that would no longer be necessary.

a combination of labour and cumulative upkeep relative to a horizon. If an optimal update is found (i.e. $\Delta^* \neq \varnothing$), it is then applied; otherwise the environment remains unchanged (i.e. $\delta^* = \delta_\varnothing$). During the application of an update, the labour costs are measured and used to update the labour estimator $\widehat{labour}$. The review interval is then succeeded by a new stable interval.

Stable interval. Once the update produced during the review interval has been performed, the stable interval begins. The purpose of the stable interval is to maintain uninterrupted streams that satisfy targets, while monitoring the upkeep of the environment to update the $\widehat{upkeep}$ estimator. The stable interval ends when one of two events occur: (1) if a short-term negative perturbation is detected, the review interval is started immediately in order to mitigate the increase in cost induced by such a perturbation; and (2) if the horizon is reached, the review interval is started as scheduled in order to check for possible improvements as the result of any long-term positive perturbations that occurred during the stable interval.

Both the review interval and the stable interval are thus responsible for monitoring observed costs.

4.3.1 Update procedure

Whenever the review interval is started, we search for and apply an optimal update if one exists. We denote δ_p to represent the perturbation that started to review cycle, if one exists; otherwise $\delta_p = \delta_\varnothing$. It is applied to a previous environment ε_{-1} to yield the current environment $\varepsilon_0 = \varepsilon_{-1} \otimes \delta_p$. The challenge is to find an optimal update δ^* to mitigate any suboptimality induced by δ_p, yielding the next environment[8] $\varepsilon_1 = \varepsilon_0 \otimes \delta^*$. This is done through a three-step approach shown below.

Exploration. The procedure for reconfiguration is shown in Algorithm 4. Nodes represent CUs-to-be that should become part of the resulting environment. The `EXPLORE` procedure first generates a root node which is a placeholder that is used to represent the targets (line 7). For example, if there are three targets, the root node will be a ternary node such that the tags for every input correspond to the tags of the targets, and the ports correspond to the desired ports of the targets. The task of `EXPLORE` is to build a valid computation graph starting from the root node. To do so, it will need to expand nodes in the graph with children satisfying that node's inputs. The combination of a node and an input index is therefore called a *job*. Jobs are kept track of as part of the $openJobs$ stack (line 3), and updated

[8]The perturbation $\varepsilon_{-1} \otimes \delta_p$ is thus similar to the game-theoretical *move by nature*.

Algorithm 4: Exploration procedure

```
1  function EXPLORE (Environment ε, ChangeSet δ_p):
2  registry ← new Map()
3  openJobs ← new Stack()
4  trace ← new Stack()
5  bestTrace ← new Stack()
6  bestCost ← ∞
7  Node root = new Node(createRoot(ε))
8  running ← true
9  while running do
10     expansionFailure ← false
11     while |openJobs| > 0 ∧ ¬expansionFailure do
12         Job job ← openJobs.pop()
13         Node next ← registry[job.tid]
14         if EXPAND (next, trace, registry, ε, δ_p, bestCost) then
15             if ¬next.virtual[job.port] then
16                 Add children to openJobs
17             end
18             Reset candIndex for all jobs in openJobs
19         else
20             expansionFailure ← true
21             openJobs.push(job)
22         end
23     end
24     if |trace > 0| then
25         (from ⇒_i to, cost) ← trace.pop()
26         if ¬expansionFailure ∧ bestCost > cost then
27             bestTrace ← trace ∪ (from ⇒_i to, cost)
28             bestCost ← cost
29         end
30         registry[from].children[i] ← nil
31         registry[from].virtual[i] ← false
32         if ⇒=→ then
33             registry[to] ← nil
34             Remove invalidated jobs from openJobs
35         end
36         openJobs.push(new Job(from, i))
37     else
38         running ← false
39     end
40 end
41 return COMPILE (bestTrace, ε)
```

when necessary. The choices made while building the graph are likewise stored in the *trace* stack (line 4).

The procedure runs by sequentially considering every job in *openJobs*

Algorithm 5: Node expansion

```
1  function EXPAND (Node node, var i, Stack trace, Map registry, Environment ε,
     ChangeSet δ_p, var bestCost) :
2  children[i] ← nil
3  virtual[i] ← false
4  expanded ← false
5  while ¬expanded ∧ candIndex[i] < numCandidates(node.tid, ε, i) do
6      candidateTID ← getCandidate(node.tid, ε, δ_p, candIndex[i])
7      (from ⇒_i to, sumCost) ← trace.peek()
8      cost ← cost(candidateTID)
9      if registry[candidateTID] = nil then
10         if sumCost + cost < bestCost then
11             Node child ← new Node(candidateTID)
12             children[i] ← child
13             virtual[i] ← false
14             registry[candidateTID] ← child
15             trace.push((tid →_i candidateTID, sumCost + cost))
16             Reset inputs succeeding i
17             expanded ← true
18         end
19     else
20         children[i] ← registry[candidateTID]
21         virtual[i] ← true
22         trace.push((tid ⇝_i candidateTID, sumCost))
23         expanded ← true
24     end
25     candIndex[i] ← candIndex[i] + 1
26 end
27 return expanded
```

and calls the EXPAND procedure on these nodes (lines 11–23). If the EXPAND procedure succeeds, any new children have their inputs added to *openJobs*. Sometimes EXPAND will find an existing node. In that case it has already been expanded as the result of the DFS approach, and does not need its inputs added as jobs. Whenever EXPAND fails, the failing job is returned to *openJobs* and backtracking is applied (lines 24–39). EXPAND can fail when all candidates for expansion have been exhausted, either due to having been attempted already, or because they result in the graph's cost exceeding the current best cost. When backtracking is performed, the last action stored in *trace* is reverted and a corresponding job as added. This will cause EXPAND to try a different candidate. For every valid graph, we check whether it is better than the currently best solution, and if so we replace it. Once no more backtracking is possible, we use the best trace and convert it into a change set using the COMPILE procedure (line 41).

Algorithm 6: Compilation procedure

```
1  function COMPILE(Stack trace, Environment ε):
2  δ ← (∅, ∅, ∅, ∅, ∅, ∅, ∅)
3  removalSet ← CU_ε
4  channelMap ← new Map()
5  cidMap ← new Map()
6  foreach (from ⇒_i to, cost) ∈ trace do
7      if to.cid ∈ removalSet then
8          removalSet \ {to.cid}
9      end
10     if ⇒ = → then
11         chan ← getUniqueID()
12         channelMap[to] ← chan
13         cid ← getUniqueID()
14         cidMap[to.tid] ← cid
15         δ ← δ ∪ spawn(ε, cid, to.tid, ∅) ∪
             connect_↓(ε, cidMap[from.tid], i, chan) ∪ connect_↑(ε, cid, chan)
16     else
17         chan ← channelMap[to]
18         δ ← δ ∪ connect_↓(ε, cidMap[from.tid], i, chan)
19     end
20 end
21 foreach cid ∈ removalSet do
22     δ ← δ ∪ destroy(ε, cid)
23 end
24 return δ
```

Expansion. The EXPAND procedure is described in greater detail in Algorithm 5. The procedure is applied to a specific node and attempts to find a valid child node for a specified input index i. The corresponding *action* taken is then added to the *trace*. Actions can represent the spawning of and connecting to new CUs ($\rightarrow$), or the reusing of nodes ($\rightsquigarrow$) that were previously added to the exploration graph as part of the current call to EXPLORE.

Every node keeps track of which candidates it has thus far considered for expansion for every input index. This is done by maintaining a $candIndex$ array of candidate indices, where each index corresponds to the next candidate to be attempted for that input index. The procedure attempts successive candidates until it either finds one that works, or runs out of candidates for input index i (lines 5–26). Specifically, the procedure considers transformations $candidateTID$ and checks if they occur in the *registry* map, which maps TIDs to nodes. If a node already exists for $candidateTID$, it is reused (lines 20–23), i.e. a virtual connection. Otherwise, a new node is created iff this does not result in the cost exceeding the best cost (lines 10–18), i.e. a new connection. Finally, the procedure returns whether expansion was successful or not (line 27).

Change set compilation. The `COMPILE` procedure described in Algorithm 6 constructs a change set δ^* from the trace produced by `EXPLORE` in conjunction with `EXPAND`. The change set is first composed of a number of spawn and connect actions. Spawn actions are called for nodes representing new CUs (line 15). A newly spawned CU then needs to be connected to a channel for its output port. The same channel is used to connect the input port of the receiving CU to. For connections to existing CUs, only the receiving CU needs to be connected with its input port. In these cases, the pre-existing channel is used to connect to. Finally, any CUs existing in the original environment that do not occur in the trace are scheduled for destruction. This ensures that CUs that are not in use do not linger and therefore do not accumulate upkeep.

To better illustrate how the three procedures interact, the following example illustrates two key scenarios. In the first, the environment is completely empty as it would be when the system is first started, and a perturbation populates the environment for the first time. The second case deals with perturbations that negatively impact the environment and which must be resolved to guarantee semantic subscriptions are maintained.

Example 12 (Finding an optimal change set)**.** *Consider for a horizon $H = 10$ an environment $\varepsilon = \langle \varnothing, F, T, \sim \rangle$ such that*

$$\begin{aligned} F = \{ & \langle tid_1, f_1(x), [A], B \rangle, \\ & \langle tid_2, f_2(x), [C], D \rangle, \\ & \langle tid_3, f_3(), [], E \rangle, \\ & \langle tid_4, f_4(), [], F \rangle, \\ & \langle tid_5, f_5(), [], G \rangle \}; \end{aligned} \tag{4.31}$$

$$\begin{aligned} T = \{ & \langle qid_1, B, 101 \rangle, \\ & \langle qid_2, D, 102 \rangle \}, \end{aligned} \tag{4.32}$$

and the similarity relation is reflexive and further includes $A \sim E$, $A \sim F$, $B \sim F$, and $B \sim G$. We thus have an environment in which no CUs are active, five transformations are registered, and two targets are registered. We will assume that the cost estimators yield 1.0 labour and 1.0 upkeep for each of the five transformations. Additionally, the perturbation is described by

$$\delta_p = (\varnothing, \varnothing, \varnothing, \varnothing, T, \varnothing), \tag{4.33}$$

meaning that the disturbance is the registration of the targets T for example by a human operator. Since δ_p is a short-term negative perturbation ($\infty > 0$), the `EXPLORE`(ε, δ_p) procedure described in Algorithm 4 is called to mitigate the perturbation-induced cost increase.

After initialising the stacks and map, the root node is created. This root node is based on the set of targets T and is represented by a placeholder transformation

$\langle root, \varnothing, [D, E], \mathsf{none} \rangle$. *The first call to* `EXPAND` *is done on this root node with an empty trace and a bestCost value corresponding to* ∞. *The expansion is performed in a depth-first manner, starting with the first input of the root node. The candidates are determined by the similarity relation* $\sim$. *Therefore, any transformations with an output tag equal to an input tag under consideration qualify as candidates for expansion. The first input tag of the root node is B, which is only equal to the output tag of transformation* tid_1. *Since* tid_1 *does not exist in the registry, it cannot be reused, so a new CU would have to be spawned from it. The total cost of such a CU would be 11.0; 1.0 from the labour and 10.0 from the upkeep over the length of the horizon. Since our current cost is 0.0, adding 11.0 would not exceed the bestCost value of* ∞, *so the candidate is used. The trace now consists of one entry;*

$$[(root \rightarrow_0 tid_1, 11.0)]. \tag{4.34}$$

The `EXPAND` *procedure is subsequently called again for the first input index of the node for* tid_1. *Its input tag corresponds to A, which is similar to the output tag of* tid_3 *and* tid_4. *Maintaining the order of transformations,* tid_3 *is chosen first, resulting in the trace*

$$\begin{aligned}[&(tid_1 \rightarrow_0 tid_3, 22.0), \\ &(root \rightarrow_0 tid_1, 11.0)].\end{aligned} \tag{4.35}$$

Since tid_3 *has no dependencies, the search continues with the second input of the root node, yielding* tid_2 *as a candidate, followed by* tid_4. *The first solution thus has a trace*

$$\begin{aligned}[&(tid_2 \rightarrow_0 tid_4, 44.0), \\ &(root \rightarrow_1 tid_2, 33.0), \\ &(tid_1 \rightarrow_0 tid_3, 22.0), \\ &(root \rightarrow_0 tid_1, 11.0)]\end{aligned} \tag{4.36}$$

and a cost of 44.0. The `EXPLORE` *procedure then starts backtracking. The trace head*

$$(tid_2 \rightarrow_0 tid_4, 44.0) \tag{4.37}$$

is first removed, and `EXPAND` *is called on its head node* tid_2 *with a best cost of 44.0. While* tid_5 *is a valid candidate, its cost would be equal or greater than the best cost of 44.0, so* `EXPAND` *returns failure and backtracking continues. The next trace head is*

$$(root \rightarrow_1 tid_2, 33.0), \tag{4.38}$$

for which there are no alternatives, so backtracking continues further. Next is trace head

$$(tid_1 \rightarrow_0 tid_3, 22.0), \tag{4.39}$$

where `EXPAND` *is called on the* tid_1 *node, which does have an alternative candidate* tid_4. *Since picking* tid_4 *would not exceed the best cost, it is picked, resulting in a*

trace

$$[\,(tid_1 \rightarrow_0 tid_4, 22.0)\,, \quad (root \rightarrow_0 tid_1, 11.0)]. \tag{4.40}$$

The `EXPAND` *procedure returns success, but* `EXPLORE` *still has the root node as an open job to reflect the backtracking which removed its subgraph at its second input, so* `EXPAND` *is called on the root node. Due to the change from tid_3 to tid_4 earlier in the trace, the root node is allowed to pick tid_2 as its candidate again. Expansion of the tid_2 node subsequently yields tid_4 and tid_5 as candidates. Adhering to the ordering, tid_4 is chosen first. This time, tid_4 already exists in the registry, so it can be reused for free, resulting in the trace*

$$[\,(tid_2 \rightsquigarrow_0 tid_4, 33.0)\,, \quad (root \rightarrow_1 tid_2, 33.0)\,, \quad (tid_1 \rightarrow_0 tid_4, 22.0)\,, \quad (root \rightarrow_0 tid_1, 11.0)] \tag{4.41}$$

and a cost of 33.0. After this point, no better solutions are found, and backtracking exhausts the trace.

The `EXPLORE` *procedure then returns the result of applying the* `COMPILE` *procedure to the trace given the environment ε. This procedure runs through the trace, starting at the bottom of the stack, choosing unique channels and CU identifiers. The first item is root $\rightarrow_0 tid_1$, which requires the spawning of a CU of type tid_1 and its subsequent connection to the desired target channel 101. A unique channel is randomly chosen for its input; we will assume it is channel 1. This is followed by the spawning of a CU of type tid_4, the output for which is connected to channel 1, and which has no inputs. Then follows the spawning of a CU of type tid_2 whose output is connected to channel 102 as determined by the second target, and whose input channel us chosen to be channel 1 as it shares the source CU of type tid_4. The resulting change set then becomes $\delta^* = (CU^+, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing)$, where*

$$CU^+ = \{\, \langle cid_1, tid_1, [1], 101, \varnothing \rangle\,, \quad \langle cid_2, tid_4, [], 1, \varnothing \rangle\,, \quad \langle cid_3, tid_2, [1], 102, \varnothing \rangle\,\}. \tag{4.42}$$

The update $\varepsilon' = \varepsilon \otimes$ `EXPLORE`(ε, δ_p) thus yields a resulting environment

$$\varepsilon' = \langle CU^+, F_\varepsilon, T_\varepsilon, \sim_\varepsilon \rangle\,. \tag{4.43}$$

When this update is performed, the estimators for $\widehat{labour}$ are updated based on the observed resource usage associated with instantiating transformations.

The update above marks the end of the review interval and the start of the stable interval. The stable interval normally has a duration equal to the horizon length,

during which the estimators for $\widehat{upkeep}$ are updates based on the observed resource usage of CUs. Short-term negative perturbations could however cut this duration short. To better illustrate the adaptivity of semantic subscriptions, we will assume that such a perturbation indeed occurs.

For the duration from the review cycle's completion to the perturbation, the targets qid_1 and qid_2 ensured that there would be a stream sent over channels 101 and 102, for which the semantics are described by the tags B and D respectively. The occurrence of the perturbation jeopardises these streams. In the worst case, no more samples are sent out on the channels, effectively freezing the streams. The premature termination of the stable interval and start of the review interval is meant to quickly mitigate this problem. We will assume that the perturbation corresponds to the crash of a CU, illustrated by

$$\delta_p = (\varnothing, \{\langle cid_3, tid_2, [1], 102, S\rangle\}, \varnothing, \varnothing, \varnothing, \varnothing). \tag{4.44}$$

The perturbation δ_p encodes the fact that the CU of type tid_2 and with identity cid_3 has been removed from the environment. This puts a hole in the computation graph, as streaming data from CU cid_2 sent over channel 1 is no longer processed, nor is the stream that would have resulted from that processing sent over channel 102 to satisfy target qid_2. Furthermore, the estimators for labour and upkeep have been updated since our previous optimal update, and the upkeep cost of CU cid_2 is determined to be 3.0 per time-unit instead of the estimated 1.0 per time-unit.

The $\texttt{EXPLORE}(\varepsilon' \otimes \delta_p, \delta_p)$ procedure is run to obtain an optimal update that will cost-efficiently resume the data stream on channel 102. The process is the same as before, except that now we can use CUs from the environment $\varepsilon'' = \varepsilon' \otimes \delta_p$, which are prioritised over spawning new CUs from transformations. One advantage of reusing CUs is that no labour cost is acrued. This leads to an initial trace

$$\begin{aligned}[&(tid_2 \rightsquigarrow_0 cid_2, 51.0),\\ &(root \rightarrow_1 tid_2, 51.0),\\ &(cid_1 \rightarrow_0 cid_2, 40.0),\\ &(root \rightarrow_0 cid_1, 10.0)]\end{aligned} \tag{4.45}$$

and a cost of 51.0; reusing cid_1 requires an upkeep of 10.0, reusing cid_2 requires an upkeep of 30.0, spawning a CU of type tid_2 requires an upkeep of 10.0 and labour equal to 1.0, and connecting this new CU to a CU we already paid for is cost-free. Unfortunately, while this solution is a quick-fix of the problem, it is not the best solution. The upkeep cost of cid_2 has increased sharply since the last review interval. Therefore, backtracking yields another solution which is optimal;

$$\begin{aligned}[&(tid_2 \rightarrow_0 tid_5, 43.0),\\ &(root \rightarrow_1 tid_2, 32.0),\\ &(cid_1 \rightarrow_0 tid_3, 21.0),\\ &(root \rightarrow_0 cid_1, 10.0)].\end{aligned} \tag{4.46}$$

This way, we no longer expend resources on the upkeep of cid_2, and the one-off labour cost is insignificant with a horizon length of 10 time-units. The EXPLORE *algorithm next calls on* COMPILE*. As before, a change set is generated which spawns and connects CUs. This time we however also destroy CU cid_2 to remove its drain on the upkeep, as it is never removed from the removalSet due to not occurring in the trace. The resulting change set is therefore $\delta^* = (\mathcal{CU}^+, \mathcal{CU}^-, \varnothing, \varnothing, \varnothing, \varnothing)$, where*

$$\begin{aligned} \mathcal{CU}^+ = \{ &\langle cid_4, tid_3, [], 1, \varnothing \rangle , \\ &\langle cid_5, tid_2, [2], 102, \varnothing \rangle , \\ &\langle cid_6, tid_5, [], 2, \varnothing \rangle \}; \end{aligned} \tag{4.47}$$

$$\mathcal{CU}^- = \{ \langle cid_2, tid_4, [], 1, S \rangle \}. \tag{4.48}$$

The application of δ^* to environment ε'' then yields a new environment

$$\varepsilon'' \otimes \delta^* = \langle \mathcal{CU}, F_\varepsilon, T_\varepsilon, \sim_\varepsilon \rangle \tag{4.49}$$

such that the new computation graph $\mathcal{CU}$ is described by

$$\begin{aligned} \mathcal{CU} = \{ &\langle cid_1, tid_1, [1], 101, S \rangle , \\ &\langle cid_4, tid_3, [], 1, \varnothing \rangle , \\ &\langle cid_5, tid_2, [2], 102, \varnothing \rangle , \\ &\langle cid_6, tid_5, [], 2, \varnothing \rangle \}. \end{aligned} \tag{4.50}$$

As can be seen from the environment $\varepsilon'' \otimes \delta^*$, we now have four CUs satisfying the two targets. This leads to the resumption of the previously-frozen stream over channel 102, fixing the problem caused by the most recent perturbation in a cost-efficient manner.

4.3.2 Correctness

The EXPLORE procedure is designed to find an optimal update if one exists, even if the original environment is invalid and therefore has a cost exceeding MAX_COST. Note that while there may exist different optimal updates with the same cost, only the first one found is selected; the others are pruned. In order to show the *correctness* of the EXPLORE procedure, it must be shown to return an optimal update.

Theorem 7 (Correctness)**.** *The* EXPLORE *procedure is* correct*, meaning that for any environment ε resulting from a perturbation δ_p, and any horizon of length H, for the set of optimal change sets Δ^* defined as*

$$\begin{aligned} \Delta^* = \arg\min_{\delta} \widehat{cost}(\varepsilon, \delta, H) \\ \text{subject to } \widehat{cost}(\varepsilon, \delta, H) \leq \texttt{MAX\_COST}, \end{aligned} \tag{4.51}$$

the following implication holds:

$$\Delta^* \neq \varnothing \rightarrow \texttt{EXPLORE}(\varepsilon, \delta_p) \in \Delta^*, \tag{4.52}$$

i.e. if some optimal change sets exist, the `EXPLORE` *procedure will return one of them.*

Proof. The proof is based on Algorithms 4, 5, and 6. In particular, it is first shown that the exploration procedure exhaustively finds all change sets δ so that $\varepsilon \otimes \delta \in \mathsf{Valid}$ if the guard $sumCost + cost < bestCost$ on line 10 in Algorithm 5 is omitted. It is then shown that the inclusion of this guard excludes suboptimal change sets, thereby returning an optimal change set if one exists.

The `EXPLORE` procedure performs a depth-first expansion of the root node when run for the first time. This sequence of operations is enforced by the stack of open jobs; whenever more expansions are available, they are pushed to the top of the stack. This means that when the stack is empty, no more expansions can be performed, and a complete computation graph has been found. The sequence of actions resulting in this graph is maintained as a trace stack. This allows us to backtrack by undoing actions and considering alternative candidates.

The candidates for expansion are kept track of within the nodes of the graph. Whenever we backtrack to a node, we increment the candidate index (*candIndex*) for the input index of interest. When a suitable alternative candidate is found, the trace is updated accordingly. Since a change has been made to the graph, this means we can reset all of the candidate indices of future jobs. This way we ensure that we find all valid change sets.

Now consider what happens if we reinstate the cost guard. For `EXPLORE` to return a suboptimal solution, either there exists no solution in Δ^* or the optimal solution is not considered in lines 9–40. The former is a contradiction with our assumption that $\Delta \neq \varnothing$. The latter can only occur if the cost guard on line 10 in `EXPAND` prunes away the optimal solution. Since the guard only excludes expansions that would lead to costs greater than the best cost, that would mean negative costs are necessary for this to occur. But negative costs are not allowed, so `EXPAND` cannot prune away the optimal solution. Therefore `EXPLORE` cannot return a suboptimal solution whenever Δ^* is non-empty. □

4.3.3 Any-time extension

The `EXPLORE` procedure quickly finds a first solution (or finds that none exist), which it subsequently improves on through an exhaustive consideration of alternatives. Alternatively, the procedure could stop considering alternatives prematurely and return the best solution found thus far. Such an any-time extension of `EXPLORE` is useful in cases where an exhaustive search would take too long and we are willing to sacrifice the optimality

of the produced change set in favour of getting a change set faster. We therefore consider a variant of EXPLORE which in addition to its usual arguments takes a value *timeout* corresponding to the time-point after which EXPLORE stops backtracking on its trace (line 24), effectively extending the guard to $|trace| > 0 \wedge runtime \geq timeout$ where *runtime* represents the number of time-units that passed since the procedure was started. The original correctness criterion can then be generalised to

$$\lim_{timeout \to \infty} \big(\texttt{EXPLORE}(\varepsilon, \delta_p, timeout)\big) \in \arg\min_{\delta} \widehat{cost}(\varepsilon, \delta, H) \quad \text{subject to } \widehat{cost}(\varepsilon, \delta, H) \leq \texttt{MAX\_COST}. \tag{4.53}$$

Given a finite value for *timeout*, it cannot be guaranteed that EXPLORE will return an optimal change set. In such a case the direction of exploration becomes a determining factor for the quality of the result. Heuristics can be used to improve the result quality by guiding the direction of exploration based on background knowledge of the search space. Specifically, the getCandidate procedure in EXPAND imposes a total ordering $\prec$ on the available candidates at every node. When the procedure plans to perform a spawn action, both labour and upkeep costs are acrued. When instead an existing CU is used, labour costs are eliminated. If a previously-expanded node can be used, all costs are eliminated. Therefore, the total order

$$\text{Spawn from TFs} \prec \text{Reuse CUs} \prec \text{Reuse nodes} \tag{4.54}$$

would prioritise cheap options before considering more expensive ones. A perturbation δ_p can provide further guidance by encoding cases in which CUs are destroyed. If the associated transformation was not removed, the ordering of candidate transformations can consider this transformation first, as it will likely provide an initial solution fast. Finally, additional heuristics taking into account properties of transformations and CUs can be considered, for example their cost, tags, or identifiers which imply freshness.

The any-time extension of EXPLORE also allows for the inclusion of its own runtime into the configuration cycle. The length of the horizon H is used to determine accumulated upkeep during the stable interval. In the any-time version, we can consider a *configuration cycle length*

$$H^+ = H_{review} + H_{stable} \tag{4.55}$$
$$= timeout + H \tag{4.56}$$

instead, which fixes the configuration cycle to a regular pattern.

4.4 Ontology-based model representation

The formal model for stream reasoning frameworks allows us to precisely describe system configurations in terms of environments, and the change

sets that can be applied to those environments. However, different realisations of this type of framework may use different internal representations. This can lead to situations wherein two different realisations based on the same formal model use two different representations. Such inconsistencies can lead to difficulties if the two are expected to interoperate.

Definition 37 (Semantic interoperability)**.** Semantic interoperability *refers to the ability to interoperate at a semantic level, even if the local representations under consideration differ.*

The Semantic Web was initially proposed by Berners-Lee et al. (2001) as an approach to making the World Wide Web machine-readable so that concepts could be formalised and exchanged, making it a good candidate to realise semantic interoperability. The Web Ontology Language (OWL) was described by the W3C in McGuinness et al. (2004), and was designed to describe such ontologies.

4.4.1 DyKnow ontology

Semantic Web technologies were used to generate a *DyKnow ontology*[9]. Ontologies in the Semantic Web are based on Description Logic (DL), which makes it possible to perform inference on them to obtain indirect knowledge. The DyKnow ontology describes the concepts presented as part of the formal model, as well as the relations that exist between these concepts. A concept hierarchy is shown in Figure 4.2, and a more detailed description of the ontology is presented in Appendix A using Manchester syntax for human readability.

The ontology formalises concepts such as CUs and the transformations they are instances of. For example, the dyknow:Transformation concept is defined in DL as

$$\begin{aligned}\text{Transformation} \sqsubseteq &\exists \text{hasName.xsd:Name} \\ &\sqcap \exists \text{hasCostModel.LabourCostModel},\end{aligned} \tag{4.57}$$

where

$$\text{LabourCostModel} \sqsubseteq \text{CostModel}. \tag{4.58}$$

dyknow:Transformation objects can further have input and output ports using the dyknow:hasInPort and dyknow:hasOutPort relations. The name of a dyknow:Transformation object then corresponds to a *tid*; the relations to dyknow:Port objects are used for $itag_1, \ldots, itag_n$ and *otag*; and the *cost* is represented by a dyknow:LabourCostModel.

CUs are also encoded in the ontology with the dyknow:ComputationUnit concept;

$$\begin{aligned}\text{ComputationUnit} \sqsubseteq &\exists \text{hasName.xsd:Name} \\ &\sqcap \exists \text{hasCostModel.UpkeepCostModel}.\end{aligned} \tag{4.59}$$

[9] Available at: `http://www.dyknow.eu/ontology/`

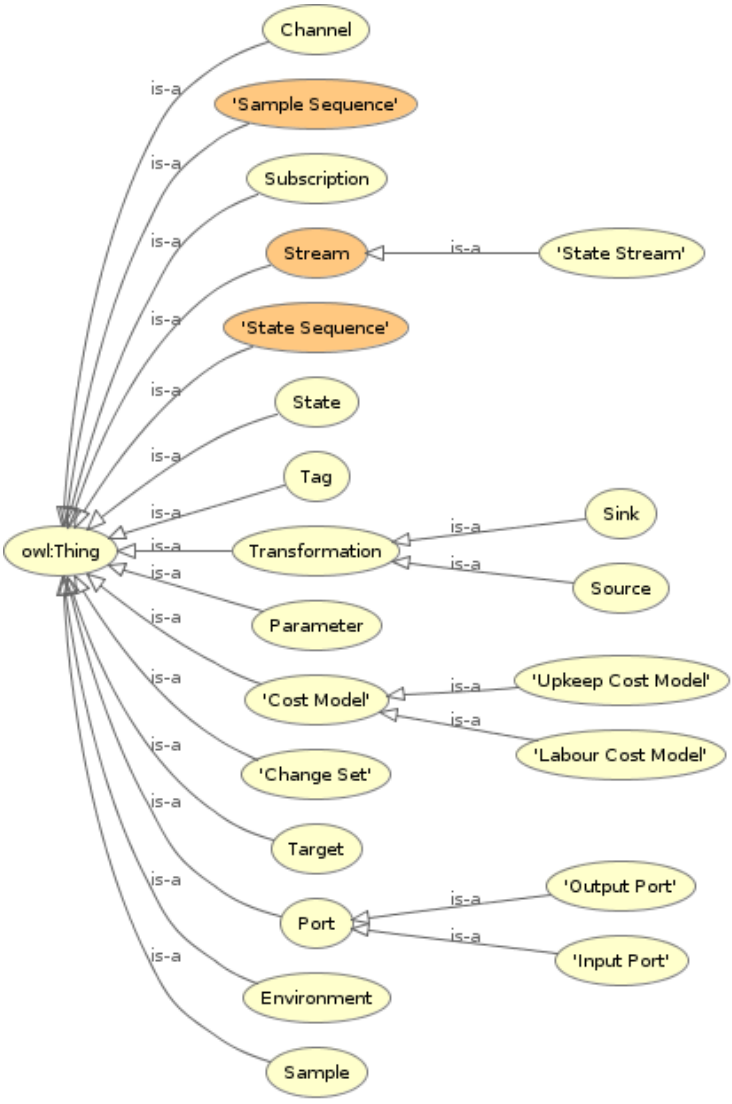

Figure 4.2: Hierarchical concept graph of the DyKnow ontology.

CUs can be connected via a dyknow:Subscription, which is defined as

$$\begin{aligned}\text{Subscription} \sqsubseteq & \exists \text{fromCU.ComputationUnit} \\ & \sqcap \exists \text{fromPort.OutPort} \\ & \sqcap \exists \text{toCU.ComputationUnit} \\ & \sqcap \exists \text{toPort.InPort} \\ & \sqcap \exists \text{hasChannel.Channel},\end{aligned} \tag{4.60}$$

meaning that a dyknow:Subscription must have some input and output port, as well as some input and output CU. Further, it is associated with a dyknow:Channel, which is used to represent the transportation mechanism over which streams can flow from CU to CU. These channels are only required to have some name, i.e.

$$\text{Channel} \sqsubseteq \exists \text{hasChannelName.xsd:string}. \tag{4.61}$$

The semantic representation thus matches the formal definition of computation graphs, and adds additional concepts (i.e. channel) that are necessary for realisations of the formal model.

Finally, targets are represented using the dyknow:Target concept;

$$\begin{aligned}\text{Target} \sqsubseteq & \exists \text{hasName.xsd:Name} \\ & \sqcap \exists \text{hasChannel.Channel} \\ & \sqcap \exists \text{hasTag.Tag}.\end{aligned} \tag{4.62}$$

Targets are thus also extended with a channel over which the resulting stream is expected. The dyknow:hasTag connects dyknow:Tag objects to a dyknow:Target object. The dyknow:Tag objects are in turn connected to semantic descriptions with the dyknow:hasTagDescription relation.

CUs, transformations and targets can be associated with dyknow:Environment objects to clearly distinguish between different environments. This makes it possible for a knowledge base to represent not just a representation of a local environment, but also that of external environments, for example on different platforms. Configuration information can further be exchanged using a common vocabulary, allowing agents to interpret configurations of other agents and to share them in a multi-agent system. Furthermore, different realisations of the formal model for stream reasoning frameworks can use and extend the ontology while retaining interoperability. For example, the dyknow:Channel concept does not specify a specific transportation mechanism.

Because ontologies in OWL are based on DL, we can apply inference to the ontological data. This makes it possible to obtain implicit information from explicit information. One example of a potentially useful property is the transitive dyknow:dependsOn object property, which is defined by

$$\text{dependsOn} \sqsubseteq \text{hasSubscription} \circ \text{fromCU}. \tag{4.63}$$

The dyknow:dependsOn relation for a given CU will connect it to all other CUs down the subscription pipeline. A reasoner can be used to infer these relations for every CU, such that the relations do not have to be provided explicitly, reducing the size of the populated ontology. This makes it possible to easily obtain for some CU all CUs it depends on, which can be useful for example when removing a CU to check for broken dependencies.

4.4.2 Ontological extensions

The DyKnow ontology thus provides a tool to support semantic interoperability between different realisations of the formal model for stream reasoning frameworks, even when these realisations make use of different internal representations of environments. A key observation is that the DyKnow ontology is designed to be extendable for purposes of realising the DyKnow model. These extensions can be performed in different ways while retaining a cross-compatible representation. One could thus see the DyKnow ontology as a *top-level ontology*. There are two sets of expected extensions for the DyKnow ontology: *system realisations* and *annotation language realisations*.

System realisations. The first category for ontological extensions deals with the realisation of the DyKnow model into a concrete system. In this case, concepts such as Channel or Transformation need an application-specific conceptualisation. These conceptualisations are more specific than the general concepts described in the DyKnow ontology. For example, while a channel is assumed to have an identifier, the DyKnow model does not put any constraints on what this identifier may look like, whereas a specific realisation might do so. Likewise, transformations may be realised as programs, resulting in more specific properties.

Annotation language realisations. The second category deals with the realisation of languages to annotate transformations or targets. These annotations are conceptualised by the DyKnow ontology using the Tag concept. A tag could be many things. For example, a tag may simply be a simple string of text, or it might be something more specific such as logical propositions or ontological concepts.

Different realisations can thus be represented using ontological extensions of the DyKnow ontology, as demonstrated later. Different realisations however still understand the high-level conceptualisations; a channel is a channel regardless of how it is implemented. This makes it possible for different realisations of DyKnow to remain compatible. While a multi-agent approach is beyond the scope of this thesis, the ontology serves as an important starting point for multi-agent support.

4.5 Open problems

- The choice of cost measures for CUs is difficult. Previous work, for example Lundh (2009), notes the same difficulties and instead simplifies the problem by assigning constant utility values. It seems more likely, however, that the cost of CUs would change based on the context of the operations. It would be interesting to see how well a predictive cost model could be learned in terms of computational resource usage, and which features would be the most informative for these predictions. While the model presented in this chapter provides a framework for using such predictions, learning good estimators is beyond the scope of this thesis.
- While the DyKnow model does consider the cost of environments, it does not consider the utility of the produced streams. In some implementations, a higher upkeep is associated with a higher-quality data stream. The representation of utility and the trade-off between cost and utility are interesting open problems.
- The presented approach allows for the configuration model to be represented relative to a Semantic Web ontology. This is done because we foresee future configurations spanning multiple agents in a multi-agent organisation, but additional work is necessary.

4.6 Summary

Reasoning over streams allows us to use crisp symbolic information to determine whether a statement in MSTL holds, which can be useful in applications such as execution monitoring. However, these symbols need to be assigned a truth value in order to be useful. State stream generation was previously discussed in terms of the combination and synchronisation of symbolic information streams. In many cases, and especially in the case of robotic systems, information enters the system at a much lower level of abstraction, for example as raw sensor observations. Generating a high-level information stream thus requires the ability to reason about one's own stream refinement capabilities. This chapter formalised the stream reasoning framework's computational environment as the DyKnow model. It does so by considering targets for formula symbols, abstract transformations, concrete CUs, and channels connecting CUs. The model can be represented relative to a Semantic Web ontology, allowing other (heterogeneous) systems to reason about a system's internal configuration. Finally, an algorithm for finding optimal updates was presented in the context of a configuration life-cycle consisting of recurring review and stable intervals.

Chapter 5

DyKnow-ROS: Putting it all together

By integrating the previously-presented techniques, an adaptive stream reasoning framework can be constructed. In this chapter an instance of such a framework called DyKnow-ROS is presented, which provides a realisation of the DyKnow model integrated with ROS. The choice of implementation is however general and could be applied to other supporting software. This chapter uses and extends materials from de Leng and Heintz (2016b), which primarily focused on extending ROS with reconfigurable subscriptions, and materials from de Leng and Heintz (2017), which focuses on semantic subscriptions for ROS. It presents an overview of the software architecture and services by providing concrete realisations of the abstract components presented previously. Finally, the framework is empirically tested to measure overhead cost resulting from the indirection induced by DyKnow-ROS.

5.1 Introduction

The DyKnow-ROS stream reasoning framework is an extension to ROS (Quigley et al., 2009), which is a popular robot middleware used frequently in both industry and academia. DyKnow-ROS is capable of reasoning about which streams to subscribe to and can reconfigure the system during run-time to for example generate streams required for spatio-temporal reasoning tasks.

ROS allows developers to write implementations as ROS *nodes*, which can communicate with each other by using *services* and *topics*. These nodes are combined into *packages*, of which many have been made publicly available. Topics can be used to connect nodes to establish a flow of information, which makes them the implementation counterpart to the concept of channels capable of transporting information streams. Topics are advertised by

publishers and can be subscribed to by other nodes using *subscribers*, such that a single topic can have multiple publishers and subscribers. Services allow nodes to advertise functionality to other nodes, which can then be requested by these nodes. Services (optionally) take a number of arguments and can (optionally) return a result to the service caller. ROS uses *Node Handles* to expose its API to developers of nodes, which packages such as *Image Transport* augment to support efficient image transportation. Where standard nodes correspond to individual processes when run, *nodelets* are run on threads within a *Nodelet Manager* node. Communication between nodelets is generally more efficient than between nodes. Further, nodes are instantiated either manually or through a launch file, whereas nodelets can also be instantiated using the Nodelet Manager's services. This makes it possible for programs to instantiate other programs at will.

In practice, most ROS-based systems rely on sometimes large collections of repeatedly nested launch files to operate. It may also be necessary for a user to run a number of launch files in a particular sequence in order for a system to function properly. It can be quite challenging to make changes to such files to accommodate new components, or to change configurations as part of a set-up phase. Clearly a lot of manual configurations may be necessary to operate a ROS-based system. This may work for small systems, but quickly becomes infeasible as systems grow to for example hundreds of robots. Depending on the application, operator errors can be very expensive. By applying the DyKnow model to ROS, DyKnow can benefit from the underlying architecture provided by ROS, while providing ROS with adaptive reconfigurability. We consider this to be an extension of core ROS features. In the performance evaluation we show that the induced overhead cost is minimal.

In this chapter, we use ROS to realise the DyKnow model and thereby realise an adaptive spatio-temporal stream reasoning application. The choice of ROS is based on the fact that it closely follows the DyKnow model, but other middleware could be used as well. The DyKnow-ROS system at times requires supporting functionality that is not the focus of this thesis. When this occurs, these functionalities are identified in the thesis and simplified placeholder solutions are used to complete the system. These functionalities are subsequently revisited in a discussion on opportunities for future work.

5.2 Architecture

DyKnow-ROS is a concrete realisation of the DyKnow model based on the ROS middleware. Figure 5.1 shows the high-level architecture of DyKnow-ROS with relevant components highlighted. In this case, all components are considered relevant, and the focus is on their integration. In particular, given a formula and a semantic interpretation of its symbols, the full system should be able to automatically generate a state stream over which

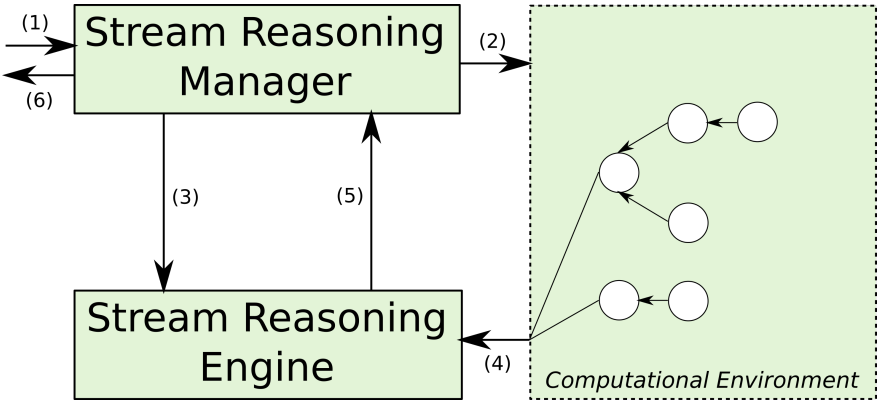

Figure 5.1: Conceptual system overview for spatio-temporal stream reasoning with adaptive state stream generation.

it then evaluates the provided formula. Once the formula has been evaluated, the computational environment should automatically be cleaned up. This combined functionality requires implementations for the stream reasoning manager, stream reasoning engine, the computational environment, and their connecting interfaces.

While ROS provides most of the transportation functionality needed to support this type of stream reasoning, its service-based interface lacks control over the way nodelets in ROS are connected. The first step towards DyKnow-ROS is therefore to extend this interface with additional services. By default, the nodelet manager provides the following services:

- **NodeletLoad:** Given a nodelet name and type, the Nodelet Manager instantiates a nodelet of that type, where the type is a reference to the nodelet's source.
- **NodeletUnload:** Given a nodelet name, the Nodelet Manager destroys that nodelet. A nodelet cannot unload itself.
- **NodeletList:** Returns an array of nodelet names.

Nodelets can thus be added, removed, and enumerated using the Nodelet Manager's services. These services are prerequisites to the *spawn* and *destroy* actions formally defined as part of the DyKnow model. Neither the Nodelet Manager nor nodelets however provide services that allow for subscribers and publishers to be changed at run-time. Developers are expected to specify configurations manually using launch files instead—ROS was not designed for the purpose of automatic (re)configuration.

To extend ROS with run-time reconfiguration services for subscribers and publishers in nodelets, different approaches could be taken. The architecture of DyKnow-ROS was chosen based on three factors: ease of

adoption, ease of use, and minimal computational overhead. DyKnow-ROS does not require a custom version of ROS, but instead provides an optional extension through the use of add-on components that build on top of nodelets. This extension is collectively referred to as a *nodelet proxy*. This allows developers to use DyKnow-ROS in some parts of their system but not others, if they so choose. Where DyKnow-ROS replaces standard ROS components, it sticks as closely to the original interface as possible and tries to limit required changes to namespace changes. This was done to make it easy to switch from standard ROS nodelets to DyKnow-ROS CUs while retaining a familiar interface. Lastly, DyKnow-ROS inevitably induces overhead computational costs by virtue of being an add-on layer on top of ROS. It seeks to keep this overhead minimal by keeping it relative to the degree of control granted to DyKnow-ROS. The more extra features from DyKnow-ROS are used, the larger the overhead.

5.2.1 The nodelet proxy

The flexibility offered by the Nodelet Manager makes it an excellent tool for dynamically reconfiguring a ROS system. As mentioned earlier, the services offered by a Nodelet Manager are limited to the loading and unloading of nodelets. DyKnow-ROS therefore complements these services with the help of persistent *nodelet proxies* that augment the ROS Node Handle. The persistent nodelet proxy is the key component that allows DyKnow-ROS to exert a greater control over the augmented nodelets, which are realisations of CUs. A developer establishes a nodelet proxy by creating a DyKnow variant of the nodelet handle instead of the usual ROS nodelet handle. Recall that the ROS nodelet handle serves as an API that can be used to call ROS functionality, such as creating publishers and subscribers. The DyKnow-ROS node handle instead delegates these calls to the nodelet proxy, which either delegates to the ROS node handle or to custom DyKnow variants depending on the functionality requested. Specifically, DyKnow-ROS provides its own publishers and subscribers that can be used in the same way as ordinary ROS publishers and subscribers. The key difference between the two lies in the indirection imposed by DyKnow-ROS. ROS publishers and subscribers connect directly to topics; a subscriber can name a topic and a callback method, whereas a publisher can name a topic and a message to be sent. The DyKnow-ROS variants instead use ports, which are in turn connected to a topic. The nodelet proxy maintains a mapping between ports and topics, and allows for this mapping to change as the result of services that are offered by the proxy. This way, ports can be associated with different topics over time, which allows for run-time reconfiguration to occur.

To illustrate the extension, a schematic of the nodelet proxy and its relation to a host nodelet is shown in Figure 5.2. The nodelet implementation by a developer is indicated by NodeletImpl, which extends ros::Nodelet. The

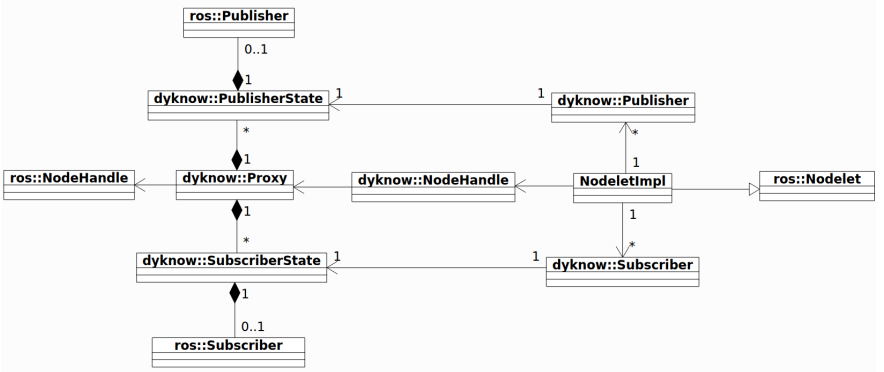

Figure 5.2: UML diagram showing the DyKnow nodelet implementation and its relation to standard ROS components.

developer is able to create a dyknow::NodeHandle, which takes a ros::NodeHandle as an argument. The dyknow::NodeHandle extends the interface provided by ros::NodeHandle, overriding some of its functionality. When a developer creates subscriptions or publishers, DyKnow-ROS provides dyknow::Subscriber and dyknow::Publisher handles. These are run-time reconfigurable version of the ros::Subscriber and ros::Publisher. A CU is also able to set a callback for whenever it is reconfigured. This can be useful when a reconfiguration requires actions to be taken by a nodelet, for example to notify some part of the system of its new subscriptions and publishers. Further, statistics such as the number of reconfigurations are maintained and made available.

The proxy adds additional services to control the mappings between topics and ports. It can also list for a given nodelet what topics are connected to which ports at the time of the service call.

- **GetConfig:** Returns a list of ports and associated topics for the nodelet the proxy is associated with.
- **SetConfig:** Takes a list of ports and topics to be connected for the nodelet the proxy is associated with.
- **GetStatistics:** Returns nodelet statistics in terms of uptime, the number of reconfigurations performed, and the number of messages sent or received for each port.

These additional services are tied to individual CUs and allow external components to keep track of and modify how CUs are connected to other CUs. With the addition of these services, the lack of configuration control is resolved.

Example 13 (A simple echo nodelet)**.** *To illustrate the subtle differences between standard ROS nodelets and DyKnow-ROS nodelets, we consider a simple*

Listing 5.1: ROS echo example

```
1 void Echo::onInit() {
2     ros::NodeHandle nh = getMTPrivateNodeHandle();
3     sub = nh.subscribe("in", 1000, &Echo::callback, this);
4     pub = nh.advertise<MessageType>("out", 1000);
5 }
6
7 void Echo::callback(const MessageType::ConstPtr& msg) {
8     pub.publish(msg);
9 }
```

Listing 5.2: DyKnow-ROS echo example

```
1 void Echo::onInit() {
2     nh = dyknow::NodeHandle(getMTPrivateNodeHandle());
3     sub = nh.subscribe("in", 1000, &Echo::callback, this);
4     pub = nh.advertise<MessageType>("out", 1000);
5 }
6
7 void Echo::callback(const MessageType::ConstPtr& msg) {
8     pub.publish(msg);
9 }
```

echo unit. Echo units can receive messages, which they then immediately forward, without performing any kind of processing on them. As such, they are one of the smallest example nodelets. Listing 5.1 shows a ROS implementation of an echo unit. We can use a local ros::NodeHandle *to create a* ros::Subscriber *and* ros::Publisher*. The subscriber is connected to the 'in' topic, with a callback to the 'callback' method. Anytime a message arrives, this method is called. Since we are using an echo unit, the message is immediately published on the 'out' topic using the publisher.*

Switching to DyKnow-ROS requires some changes as shown in Listing 5.2. Creating a dyknow::NodeHandle *results in the creation of a proxy behind the scenes. When all node handles go out of scope, so does the proxy, so we store the node handle as a member variable. The reason for requiring the proxy to be persistent is because it hosts the reconfiguration services—if it goes out of scope, the services become unavailable. The remainder of the code is the same, although instead of ROS subscribers and publishers, we get a* dyknow::Subscriber *and a* dyknow::Publisher*. The subscriber uses the 'in' port; we do not control what topic it is connected to. The same holds for the producer, which is connected to the 'out' port.*

The difference between the two code snippets is thus minimal from the perspective of the developer. However, while the syntax is largely the same, the semantics have slightly changed. As always, a developer should be aware of these underlying mechanisms.

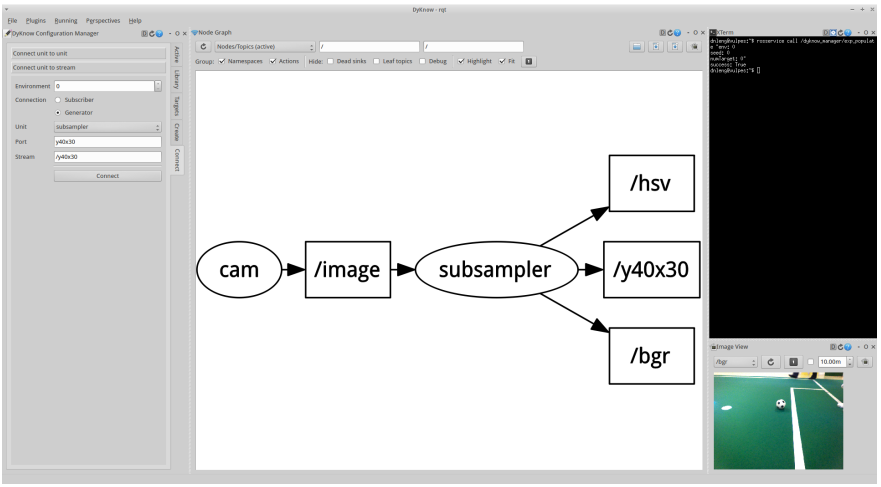

Figure 5.3: Screenshot of the interactive visualisation tool.

5.2.2 Interactive visualisation

ROS provides a wide array of visualisation tools using a Qt-based framework. For the visualisation of nodes and topics, rqt_graph provides a graphical user interface that communicates with the ROS master and produces a DOT graph. While this approach works great for nodes, it fails to detect nodelets as they are threads within the Nodelet Manager node. We therefore forked rqt_graph and replaced the communication with the ROS master to instead query the stream reasoning manager in DyKnow-ROS for its configuration model. Since ROS does not take run-time reconfiguration into consideration, we also had to switch from the manual refresh in rqt_graph to a frequency-based refresh. This was combined with a control widget to allow a user to interact with the stream reasoning manager.

A screenshot of the tool at work is shown in Figure 5.3, where the bottom left camera view was produced by the rqt_image_view widget. The graph shown in the centre panel was created using the control panel on the left. Ovals in the centre graph correspond to nodelets, and rectangles correspond to topics. The bottom-right image view shows the colour video stream. Since no changes were made to the rqt_graph interface itself, this representation is natural to ROS developers.

The control panel on the left supports a number of features based on the services provided by DyKnow-ROS. The *active* tab lists the currently active CUs together with their associated transformations, the number of input ports, and the number of output ports. A *library* tab offers a listing of transformation specifications by label, and allows a user to import or delete transformations. CUs can be instantiated through the panel as well with the *create* panel; the user provides a name for the nodelet to be created

and a type in terms of transformation specifications. The panel shown is the *connect* panel, where either a combination of two nodelets and ports are selected to be connected with a topic decided by the tool, or a single port and topic can be connected where the user gets to specify the topic name manually.

The visualisation tool gives access to all of the stream reasoning manager's services, offering an interface that can be managed by human operators. As a result, it offers the functionality expected by ROS developers as well as some extra control over the configuration during run-time.

5.3 Management of stream processing

The stream reasoning manager is responsible for setting up and maintaining configurations in support of stream reasoning. As illustrated in Figure 5.1, it interacts with the stream reasoning engine and the computational environment. It is implemented in DyKnow-ROS as a node, as is the stream reasoning engine. The manager can interact with the computational environment with the help of the proxy services. Likewise, the stream reasoning engine provides services which are presented later. Both sets of services are used by the manager, which in turn provides its own set of services acting as a client-facing interface. The services provided by the stream reasoning manager are:

- **AddTarget:** Given a target specification, store the specification under the associated label. Specifications can be overridden.
- **RemoveTarget:** Given a label, remove the target specification with that label, if any.
- **AddTransformation:** Given a transformation specification, store the specification under the associated label. Specifications can be overridden.
- **RemoveTransformation:** Given a label, remove the transformation specification with that label, if any.
- **Spawn:** Given a transformation label and name, instantiate a nodelet of that transformation type with the supplied name. Nodelets can be protected from unloading. Uses NodeletLoad.
- **Destroy:** Given a name, destroy the nodelet with that name if it exists and if it is not protected. An unprotected nodelet can destroy itself this way. Uses NodeletUnload.
- **GetModel:** Returns a listing of all running DyKnow nodelets and their port-topic connections. Also returns all stored transformation specifications. The format can be DyKnow or OWL.

The manager thus provides supporting services for changing configurations as well as acquiring a representation of the current environment. The latter service is useful for taking configuration snapshots, for example for the purpose of representing the environment in a client, as is done by the visualisation tool mentioned previously.

We can subdivide the tasks of the stream reasoning manager into two parts. The first is to keep track of the environment, i.e. what its current state is, what TFs exist, what CUs exist, etc. This basically boils down to a storage task. The second is to enforce the configuration life cycle, by regularly updating the configuration. This is the daemon component of the manager. We consider both tasks in more detail.

5.3.1 Representation of configurations

The stream reasoning manager keeps track of the state of the computational environment and provides services that can be used to change this environment. The DyKnow model specifies an ontology for representing an environment with a well-structured grammar. DyKnow-ROS makes use of this ontology to not just represent the environment, but also as a grammar for specifications of transformations and targets. Since DyKnow-ROS is a concrete realisation of the DyKnow model, it extends the DyKnow ontology to capture ROS-specifics. For example, whereas the DyKnow ontology uses the Channel concept, DyKnow-ROS refers to the Topic concept. The latter is a specialisation of the former, and enforces a well-defined grammar for topics as defined by the ROS specifications. Figure 4.2 in the previous chapter illustrated the concept hierarchy of the DyKnow ontology, which is listed in Appendix A. We briefly consider its extension to DyKnow-ROS here.

Service calls to the AddTransformation service provided by the stream reasoning manager require a uniquely-labeled transformation specification. Recall that the dyknow:Transformation concept is defined in DL as

$$\begin{aligned} \mathsf{Transformation} \sqsubseteq\ & \exists \mathsf{hasName.xsd{:}Name} \\ & \sqcap \exists \mathsf{hasCostModel.LabourCostModel}, \end{aligned} \tag{5.1}$$

where

$$\mathsf{LabourCostModel} \sqsubseteq \mathsf{CostModel}. \tag{5.2}$$

DyKnow-ROS uses the specialised concept dyknowros:ROSTransformation such that ROSTransformation $\sqsubseteq$ Transformation. Concretely, the ROSTransformation is defined in DL as

$$\begin{aligned} \mathsf{ROSTransformation} \sqsubseteq\ & \mathsf{Transformation} \\ & \sqcap =1\ \mathsf{hasSource.xsd{:}anyURI} \\ & \sqcap \exists \mathsf{hasPort.Port}, \end{aligned} \tag{5.3}$$

where

$$\mathsf{hasInPort} \sqsubseteq \mathsf{hasPort}, \tag{5.4}$$

$$\mathsf{hasOutPort} \sqsubseteq \mathsf{hasPort}. \tag{5.5}$$

This means that a ROSTransformation has at least one port, either an input or an output port. Furthermore, it has exactly one source, which is represented by a URI to a nodelet binary. This makes it possible for the manager to dynamically load specific nodelet implementations. Lastly, ports can be annotated with tags describing the semantics of the data flowing through those ports and the channels they connect to. Listing 5.3 shows an example of a transformation specification in DyKnow-ROS.

Listing 5.3: Example transformation specification in Turtle syntax

```
:undistort a :ROSTransformation ;
    :hasType "nodelet" ;
    :hasSource "package/Undistort" ;
    :hasParam [
        a :Parameter ;
        :hasName "configPath" ;
        :hasType "string" ;
        :hasValue "/path/to/configuration/cam1/" .
    ] ;
    :hasPort [
        a :InPort ;
        :hasName "rawCamera" ;
        :hasTag [
            a :Tag ;
            :hasValue "RawRGB(cam1)" .
        ] .
    ] ;
    :hasPort [
        a :OutPort ;
        :hasName "undist" ;
        :hasTag [
            a :Tag ;
            :hasValue "Undistorted(cam1)" .
        ] .
    ] ;
    rdfs:label "Undistort(cam1)" .
```

Service calls to the AddTarget service require a target specification. Similar to transformation specifications, target specifications make use of the dyknow:Target concept extended to dyknowros:ROSTarget for ROS. Targets are composed of a label, channel, and tag. In the case of ROS, the channel corresponds to a topic, i.e.

$$\mathsf{ROSTarget} \sqsubseteq \mathsf{Target} \sqcap \exists \mathsf{hasTopic}.\mathsf{Topic}. \tag{5.6}$$

where dyknowros:Topic is a specialisation of dyknow:Channel. Topics use a standardised naming convention enforced by ROS which is similar to the way paths are represented in Unix-based systems. Listing 5.4 shows an example target specification in DyKnow-ROS.

Listing 5.4: Example target specification in Turtle syntax

```
:undistortSub a :ROSTarget ;
    :hasTopicName "/result"^^rosTopic ;
    :hasTag [
        a :Tag ;
        :hasValue "Undistorted(cam1)" .
    ] .
    rdfs:label "undistortSub" .
```

This leaves us with CUs. While the service calls do not require CUs as arguments, some do return them as part of the service response. Therefore, they too have a DyKnow-ROS specification. Since nodelets act as CUs in DyKnow-ROS, we simply get

$$\mathsf{Nodelet} \sqsubseteq \mathsf{ComputationUnit} \tag{5.7}$$

for the sake of completeness.

Example 14 (Transformation and target specifications in DyKnow-ROS)**.** *Consider a smart lab equipped with four similar ceiling cameras. In this example, the cameras are using fish-eye lenses, and their positions allow them to cover most of the lab's ground surface area with some overlap. A transformation could be applied to the image streams from the cameras if they are first undistorted.*

A subscription to an undistorted stream from a camera called 'cam1' is illustrated in Listing 5.4 as a target specification. The target is labelled undistortSub and represents the desire for a stream to be produced on the /result topic with the semantic description Undistorted(cam1). The semantic description is part of the tagging language and intended to represent an undistorted image stream originating from the 'cam1' camera.

A transformation that might produce a suitable stream is illustrated in Listing 5.3. It represents a nodelet for which the binary is referred to in package/Undistort. Note that this binary makes no reference to a particular camera. This is because the transformation combined the binary with a configuration for the 'cam1' camera. It does so by providing the binary with a 'configPath' parameter, which the binary understands to be the location of a lens model file specific to the 'cam1' camera. Consequently, the transformation is labelled Undistort(cam1) to illustrate it is specific to the 'cam1' camera eventhough the binary it uses is not. The transformation has two ports; one input port expecting raw RGB images from the 'cam1' camera, and one output port producing undistorted versions of those images. The transformation is therefore a suitable candidate for satisfying the target, assuming that its dependency on raw RGB images can be resolved.

Both transformations and targets make use of tags, which are used for semantic annotations. The focus of this thesis work was however not on the development of an annotation language, but rather how one could be used in the context of the DyKnow model. Despite this, a tagging language is necessary for the DyKnow-ROS system to work. Therefore a simple tagging language is used when tags are explicitly specified, with the understanding that a better tagging language can replace this placeholder language in the future.

5.3.2 Configuration life-cycle daemon

The second task of the stream reasoning manager is to act as a daemon by reconfiguring the DyKnow-ROS configuration in accordance with the configuration life-cycle. This requires the realisation of Algorithm 4 from the previous chapter in a ROS environment. Additionally, functionality is needed for the observation of computational resource usage and its effect on the cost estimators.

The `EXPLORE` procedure and its dependencies take a computational environment ε and perturbation δ_p, and subsequently construct an optimal change set δ^* with the help of the *spawn*, $connect_{\downarrow}$, $connect_{\uparrow}$ and *destroy* actions. In DyKnow-ROS, the environment is available and monitored by the stream reasoning manager. Perturbations can be detected as well through the service calls that would qualify as perturbations. For example, if a target is added, this is achieved to a service call to the manager, thereby implicitly notifying the manager of a perturbation. As such, ε and δ^* have counterparts in DyKnow-ROS. The same holds for the aforementioned actions which make up δ^*, as each of them can be performed using the services available to the stream reasoning manager. The application of an optimal change set is then equivalent to a sequence of service calls. The `EXPLORE` procedure is deviated from in two ways. Firstly, since ROS allows nodelets to have multiple ros::Producer instances, a CU can have multiple outputs. In such a case, the CU is said to be an instance of multiple transformations which take the same inputs but produce different outputs. Secondly, CUs can be designated 'protected', in which case they are never destroyed by a resulting change set. This will cause the procedure to find a best change set given that the protected CUs are kept around.

Computational resource usage is measured in terms of CPU time, specifically by combining `utime` and `stime`. Both values are obtained by reading from `/proc/$tid/statm` on Linux for the corresponding thread identifier. For labour, we measure the CPU time associated with creating new CUs. For upkeep, the CPU time per wall time minute is accumulated by measuring the CPU times of individual callbacks from dyknow::Subscriber and dyknow::Timer objects. The former is useful for CUs that react to new inputs, whereas the other is useful for CUs that run at specific time intervals. Upkeep is measured relative to a transformations, so if multiple CUs exist for the same transformation, the upkeep would be the average CPU time measured over all of those CUs.

Finally, the estimators need to be updated using these observations. Since the focus of this thesis was not on precise estimations, a simple placeholder was chosen to fulfil the requirement of having estimators. Future work could produce better models of CPU usage. In the current state, the predicted CPU usage is simply an average over the observed CPU usage.

The cost models for labour and upkeep are referred to as

$$\mathsf{LabourCostModel}(\mathsf{historicalAverageLabour}), \tag{5.8}$$
$$\mathsf{UpkeepCostModel}(\mathsf{historicalAverageUpkeep}), \tag{5.9}$$

in the DyKnow-ROS ontology extension.

5.4 Spatio-temporal stream reasoning support

Reasoning over streams is performed by the *stream reasoning engine* in DyKnow. This composite component takes as its input formulas, state streams, and grounding information for the purpose of progression of the formulas over the provided data. Formulas can be part of a *formula group*, which represents a collection of formulas that are to be evaluated simultaneously over a single state stream. The connecting information grounds the symbols in logical formulas to specific values in the state stream. The engine produces the corresponding truth values of the formulas, assuming the reasoning procedure terminates—while progression is guaranteed to terminate, it may never rewrite a formula into a single truth value in cases where temporal operator intervals are unbounded.

In the simplest case, symbols in formulas can be directly connected to values in a state stream originating from the computational environment. However, in more complex cases, the stream reasoning engine must infer truth values associated with (some of) a formula's predicates. In this thesis, the stream reasoning engine is therefore composed of a progressor as well as a qualitative spatial reasoning engine, where the latter is a modified version of GQR. The modifications make it possible for the progressor to use GQR through ROS services. The stream reasoning engine itself provides a number of services that control the evaluation of formulas, as shown below.

- **CreateGroup:** Creates a formula group with an optionally provided label and result topic. If no label or result topic are provided, DyKnow generates them instead.
- **DestroyGroup:** Destroys a formula group by its label. This stops the progression of formulas in the group.
- **StartGroup:** Activates progression for a formula group identified by name.
- **StopGroup:** Stops the progression for a formula group by name. Cannot be resumed.
- **AddFormula:** Adds a provided formula to a formula group identified by its label. Yields an identifier (index) for the formula.

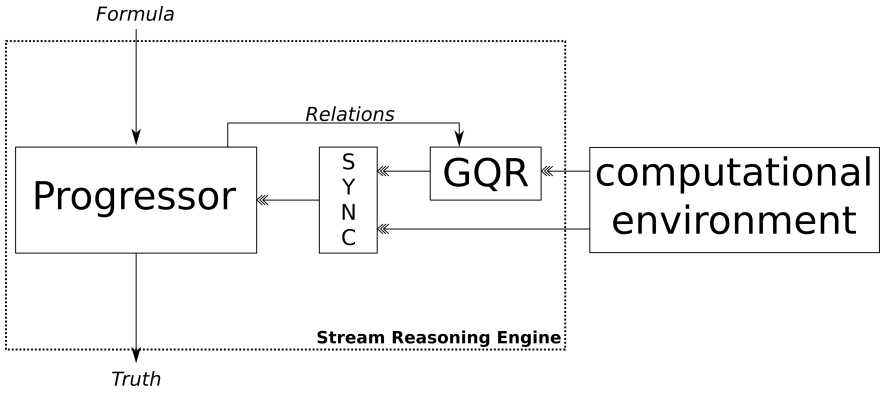

Figure 5.4: Architecture of the stream reasoning engine component.

- **RemoveFormula:** Removes a formula from a formula group identified by the group's label and formula's index. This can be done while a formula is being progressed.

Figure 5.4 illustrates the architecture of the stream reasoning engine and its relation to the generation of a state stream. State streams are needed to evaluate formulas by the progressor. They are produced for each of the formula groups, which make use of a number of symbols that are grounded in streams. These streams are acquired through the use of targets in the computational environment. The streams that are produced as a result of the targets are subsequently synchronised. Because of strict typing constraints, the synchronisation process first applies *type flattening*, in which the names, types, and values of data samples are represented in string format. The correct conversion to and from this flattened format is obtained through code generation from ROS message specifications, which basically represent the data types used for topics. The streams produced as the result of targets are thus converted into a flattened representation before being synchronised. The synchronisation method used was proposed by Heintz (2009) and makes use of the stream frequencies to determine whether and when to wait for samples. The result of the synchronisation method is a state stream in which every symbol has a corresponding value. Every formula group requires one such state stream. Once a formula group is destroyed, the targets associated with it are removed as well, unless they are shared by another formula group.

Concretely, a wff in MSTL is made up of predicates for which the truth value needs to be provided in order to run the progression procedure described in Algorithm 3. If nothing is known about the predicates by DyKnow-ROS, these truth values are expected to be provided directly in a state stream. However, if the predicate is a comparator $\oplus \in \{=, \neq, <, >, \leq, \geq\}$, the wff will contain statements of the form $x \oplus y$, where x and y can be

constants or functions ranging over objects. For example, a statement such as

$$\mathsf{altitude}(\mathsf{uav}) > 10 \tag{5.10}$$

is a predicate for which the $\mathsf{altitude}(\mathsf{uav})$ term is provided externally in a state stream. Similarly, a statement such as

$$\mathsf{altitude}(\mathsf{uav}_1) > \mathsf{altitude}(\mathsf{uav}_2) \tag{5.11}$$

is a predicate for which both terms are provided externally in a state stream. When comparators are used, a target is required for every externally-provided term, where the resulting stream provides an interpretation of the functions at different time-points. This allows the progressor to interpret the comparators.

Consider a formula group consisting of a single MSTL formula stating that all UAVs always have an altitude exceeding 50 metres, i.e.

$$\forall x \in \mathsf{UAV}\,[\Box(\mathsf{altitude}(x) > 50)]\,, \tag{5.12}$$

where the domain $\mathsf{UAV} = \{\mathsf{uav}_1, \mathsf{uav}_2\}$. The first step is for a client (human or otherwise) to generate two targets; one for $\mathsf{altitude}(\mathsf{uav}_1)$ and one for $\mathsf{altitude}(\mathsf{uav}_2)$. These transformations are annotated with tags corresponding to altitude information for the two UAVs. These tags could correspond to the functions $\mathsf{altitude}(\mathsf{uav}_1)$ and $\mathsf{altitude}(\mathsf{uav}_2)$ depending on the choice of tagging language. This results in two streams containing altitude information for the two UAVs. The second step is for the system to synchronise these two streams into a single type-flattened state stream. This then results in a new synchronised stream for which the data type is understood by the stream reasoning engine, and every sample of which contains values for the labels $\mathsf{altitude}(\mathsf{uav}_1)$ and $\mathsf{altitude}(\mathsf{uav}_2)$.

A different situation occurs for predicates from the $\mathcal{R}_8$ set of RCC-8 spatial relations. A spatial relation $R(x, y)$ is true, false, or unknown depending on other spatial relations. Since the handling of RCC-8 is part of the DyKnow-ROS architecture itself, the state stream generation queries GQR for relations $R(x, y)$ and subsequently uses that information directly. Consequently operators are required to specify targets that connect GQR to qualitative spatial relation information produced in the computational environment. Methods towards automating this connection through targets are beyond the scope of this thesis. An *ad-hoc* solution is to use RCC-8 tags to identify any transformations that produce RCC-8 information. Whenever a formula contains such a relation, DyKnow-ROS will first try to find a transformation specifically generating that relation. If none exist, it will instead use all RCC-8 relation-producing transformations, after which GQR infers further spatial relations.

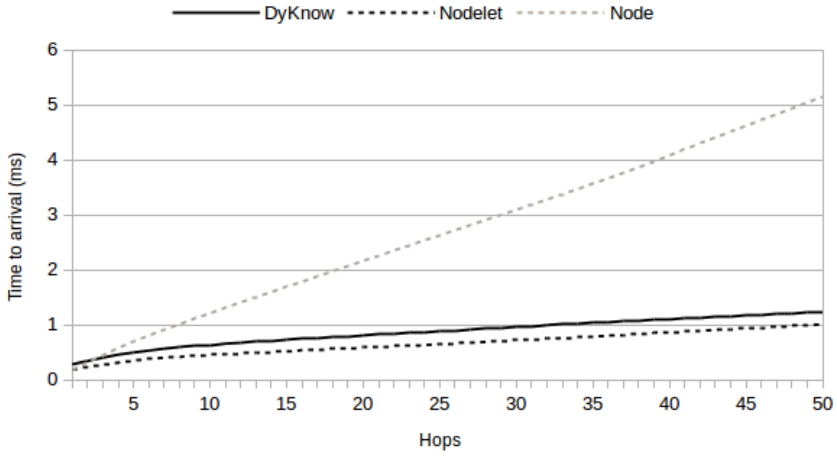

Figure 5.5: Performance graph showing the different time-to-arrivals for messages relative to the number of hops for a linear chain.

5.5 Performance evaluation

The proxy introduced by DyKnow-ROS potentially introduces an overhead in throughput. Measuring the overhead gives insights into the cost of adopting DyKnow-ROS.

Topic-based communication between nodelets is assumed to be faster than between nodes because nodelets are part of the same process and nodes are not. In this experiment, we use both as benchmarks for comparison. The computation graph is a linear sequence of connected node(let)s such that each intermediate node(let) receives from a predecessor node(let) and immediately publishes to a successor node(let). The source produces messages containing current time-stamps at a fixed frequency f. Every (intermediate) receiver checks that time-stamp against the arrival time and reports the time difference. The number of node(let)s n then corresponds to the number of message hops.

The performance results are shown in Figure 5.5, where the performance graph contrasts the number of hops to the average time-to-arrival for messages sent along the node(let) chain. The source produced 1,000 time-stamped messages at a frequency of $f = 5$Hz, which every receiver compared to the local time upon arrival prior to forwarding the message. The graph illustrates the time results for DyKnow-ROS nodelets and ROS nodelets, as well as ROS nodes. As expected, nodes are much slower than nodelets because they have to communicate between processes. The results for nodes put into perspective the overhead we can see for DyKnow-ROS nodelets when compared against standard ROS nodelets, which grows slowly

to about 0.2ms after $n = 50$ hops. We therefore conclude that the overhead induced by DyKnow-ROS is negligible.

5.6 Open problems

- DyKnow-ROS relies on nodelets for dynamic instantiation of CUs. This presents some practical problems. First, this excludes ROS nodes, since these can only be started by command-line or via `roslaunch`. Currently, node-based implementations have to be converted to nodelets, although many support both types. The second issue is that a crash of a nodelet brings down the nodelet manager, and thereby all CUs that are running as part of that nodelet manager. This means that many if not all CUs crash if one does, and recovery then requires a new nodelet manager process to be started. Some additional engineering efforts are needed to resolve these practical issues.
- ROS has some known shortcomings in terms of communication guarantees, making it less useful for real-time applications. A new version of ROS, going by the name ROS2, is under development. It would be interesting to see how ROS2 could be combined with the DyKnow model for a potential DyKnow-ROS2 realisation with real-time guarantees.

5.7 Summary

This chapter presented a realisation of the DyKnow model with the Robot Operating System (ROS), resulting in the DyKnow-ROS system. First an extension of the services provided by ROS is presented to support the DyKnow model. For practicality, a common ROS visualisation tool was also adapted to facilitate human interaction with the system. This was followed by a concrete representation of entities in the DyKnow model with the help of the DyKnow ontology extended for ROS. The configuration life cycle was realised by a daemon that uses CPU time as the computational resource of choice in DyKnow-ROS, and some simple estimators for labour and upkeep were presented. The generation of state streams with the help of targets was explained. Finally, the overhead induced by the extensions to ROS was measured and shown to be negligible.

Chapter 6

Case study

One of the thesis contributions—integration—deals with the applicability and practicality of the suggested solutions. The previous chapters focused first on the theoretical contributions within the two strands, followed by the engineering contributions that resulted in the DyKnow-ROS stream reasoning framework. As per the methodology discussion, there is an additional deployment step. DyKnow-ROS was therefore deployed on two SoftBank Robotics NAO robots as part of a case study that highlights the applicability and practicality of the proposed solutions. Since the framework makes use of CUs, software written by the Linköping University RoboCup team for the Standard Platform League was integrated into DyKnow-ROS to support real-world transformations.

6.1 Introduction

Our case study focuses on two NAO robots, called *Piff* and *Puff* (Swedish for *Chip 'n Dale*). Both Piff and Puff are capable of running a processing pipeline that takes in sensor information and produces ball coordinates relative to the soccer field. For the case study, we were interested in situations where semantic subscriptions could provide added value to Piff in performing its task of tracking the ball. We consider two cases; 1) Piff is tracking the ball but something goes wrong; and 2) Piff is tracking the ball and Puff offers to help for a while. Piff and Puff are assumed to be part of the same computational environment; a multi-agent system approach is beyond the scope of this thesis.

6.2 Experimental set-up

The operational environment is provided by a humanoid lab at Linköping University, which is organised to support software development for NAO

Figure 6.1: Humanoid lab (left) equipped with four ceiling cameras (right).

robots.

6.2.1 Humanoid lab

The humanoid lab is equipped with a green felt RoboCup soccer field as shown in Figure 6.1 on the left. As shown on the right, there are four cameras attached to the ceiling over the field. The ceiling cameras are AXIS M3005-V network cameras with a 118° angle of view, producing 1920 $\times$ 1080 images. These images can be used for accurate positioning of objects on the field. The coordinate system uses one of the field corners as its origin, relative to which the coordinates of other objects such as balls or robots are determined.

6.2.2 Piff and Puff

Piff and Puff are Softbank Robotics NAO humanoid robot platforms, of which an example is shown in Figure 6.2. Standing upright, they are 58cm tall and weigh 5.4kg. With normal use, the battery provides 90 minutes of autonomy. The head houses two HD cameras producing 1280 $\times$ 960 images at 30 FPS in YUV422 colour space. One is located in the forehead and faces forward; the other is located in the 'mouth' area and faces downwards. The various joints provide pose information to the system through joint position sensors. The NAO comes equipped with an Intel Atom Z530 processor running at 1.6GHz, with 1GB of RAM, 2GB of Flash memory, and an 8GB Micro SDHC. The system runs the Ubuntu 14.04 LTS operating system with the NAOqi programming framework. Our NAO platforms use a publically-available ROS driver[10] exposing the NAOqi API through ROS. More technical details can be found in the NAO technical guide[11].

[10]The NAO packages for ROS are documented at `http://wiki.ros.org/nao` (Last accessed: May 30th, 2017)

[11]The NAO technical documentation is available at `http://doc.aldebaran.com` (Last accessed: May 30th, 2017)

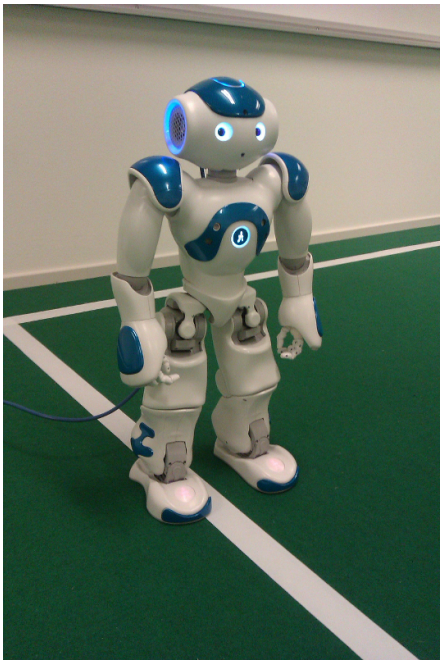

Figure 6.2: A SoftBank Robotics NAO V4 robot.

6.3 Recovery from failures

We start with a scenario in which a user wants to perform perimeter monitoring. That is, check whether a ball 'breaches' the perimeter indicated by the centre circle on the football field and, if so, notify the user. While a toy scenario, it allows us to consider the full DyKnow-ROS system's operations. The user makes no assumptions about what equipment exists in the system; only that there is a DyKnow-ROS instance which he or she is able to interact with. Neither does the user make any assumptions about the availability of equipment over time. In a sense this is a natural behaviour for a non-expert user. The user queries a system's services and the system tries to meet the user's expectations to the best of its abilities. The system further seeks to minimise the cost it incurs while satisfying the user's needs. The first step for a user is then to describe those needs in some language expression. DyKnow-ROS uses MSTL for this purpose, so the user's inquiry is described by a wff

$$\Box_{[0,1440]} \left[\mathsf{InsideCircle}(\mathsf{ball})\right], \tag{6.1}$$

meaning that for the next 24 hours (measured in minutes), the ball will remain within the circle[12]. The statement is not intended to enforce a par-

[12] Alternatively, qualitative spatial relations NTPP and TPP could be used.

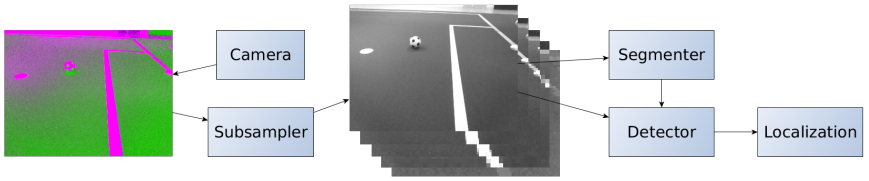

Figure 6.3: Piff and Puff's transformation pipeline conceptually showing the transformations from camera images to ball positions.

ticular situation, but rather specifies what is expected to happen. It might be the case that the ball leaves the circle for whatever reason, which should then result in the statement being evaluated to false. However, in order to determine whether the statement is true or false (or even unknown), a state stream is needed over which it can be evaluated.

The task of generating a state stream starts with the specification of targets. Recall that targets are composed of a channel identifier and a tag describing the semantics of the sought-after streaming data. The target should thus reflect that we require information on the truth value of $\mathsf{InsideCircle(ball)}$, which is a predicate. The target is illustrated in Listing 6.1.

Listing 6.1: Target specification for $\mathsf{InsideCircle(ball)}$ information

```
:target1 a :ROSTarget ;
    :hasTopicName "/target1"^^rosTopic ;
    :hasTag [
        a :Tag ;
        :hasValue "InsideCircle(ball)" .
    ] .
    rdfs:label "InsideCircle(ball)" .
```

After adding this target to the environment ε, it looks like

$$\langle \varnothing, \varnothing, \{\langle \mathsf{target1}, \mathsf{InsideCircle(ball)}, /\mathtt{target\_1}\rangle\}, = \rangle . \tag{6.2}$$

This represents a perturbation, so the environment tries to reconfigure itself. However, since no transformations exist, no solutions are found yet, and no state stream is generated. We can solve this by considering the situation wherein Piff registers its transformations to the environment. Table 6.1 shows the semantics of the set of transformations provided by the NAO robot using a short-hand notation.

The bottom_cam TF provides a YUV image stream, which can be subscribed to by the subsampler TF. This transformation down-samples the resolution of the three channels into 640x480, 320x240, 160x120, 80x60, and 40x30. The segmenter TF instances may subscribe to low-resolution Y and V channels to determine the convex hull of the green field, ignoring the space in the image which captures things outside of the field. This convex hull is combined with the Y channel by the ball_detector TF to produce pixel coordinates of balls, which is then combined with pose information by the

TID	TF label		Tags
tid_1	pose(piff)		$\varnothing$
		$\Rightarrow$	pose(piff)
tid_2	bottom_cam(piff)		$\varnothing$
		$\Rightarrow$	yuvImage(piff)
tid_3	subsampler(piff)		yuvImage(piff)
		$\Rightarrow$	imageScalePyramid(piff)
tid_4	segmenter(piff)		imageScalePyramid(piff)
		$\Rightarrow$	convHull(piff, field)
tid_5	ball_detector(piff)		convHull(piff, field), imageScalePyramid(piff)
		$\Rightarrow$	pixelPos(piff, ball)
tid_6	ball_localization(piff)		pixelPos(piff, ball), pose(piff)
		$\Rightarrow$	position(ball)
tid_7	circle_monitor(ball)		position(ball)
		$\Rightarrow$	InsideCircle(ball)

Table 6.1: Piff's transformations and their tags denoted by **itag** $\Rightarrow$ *otag*.

ball_localization TF to produce ball position data, which matches the query. Since the validity matrix is updated when transformations are added or removed, the result is a 11×19 validity matrix for the 11 inputs and 19 outputs.

As the result of the perturbation, the stream reasoning manager searches for an optimal configuration and finds one as shown conceptually in Figure 6.3. The associated change set δ is the instantiation of all transformations, and the connection of the resulting CUs in accordance with their annotations. The new environment ε is described by

$$\langle CU, F, \{\langle \text{target1}, \text{InsideCircle}(\text{ball}), /\texttt{target\_1}\rangle\}, =\rangle, \tag{6.3}$$

where the set of transformations F remains unchanged, and the set of CUs is described by

$$\begin{aligned} CU = \{ & \langle cid_1, tid_7, [/\texttt{topic\_1}], /\texttt{target\_1}\rangle, \\ & \langle cid_2, tid_6, [/\texttt{topic\_2}, /\texttt{topic\_6}], /\texttt{topic\_1}\rangle, \\ & \langle cid_3, tid_5, [/\texttt{topic\_3}, /\texttt{topic\_4}], /\texttt{topic\_2}\rangle, \\ & \langle cid_4, tid_4, [/\texttt{topic\_4}], /\texttt{topic\_3}\rangle, \\ & \langle cid_5, tid_3, [/\texttt{topic\_5}], /\texttt{topic\_4}\rangle, \\ & \langle cid_6, tid_2, [], /\texttt{topic\_5}\rangle, \\ & \langle cid_7, tid_1, [], /\texttt{topic\_6}\rangle\}. \end{aligned} \tag{6.4}$$

Piff now produces a ball position stream on the `/topic_1` topic, which can be used by the circle monitor TF to determine whether the ball is inside

the circle. This Boolean information is then transmitted on topic `target_1` as specified by the target target1. The new environment results in a stream containing the information needed for interpreting the symbols of the MSTL formula, which is synchronised, flattened, and connected to the stream reasoning engine. The progression procedure now uses the resulting state stream to incrementally evaluate the formula through rewritings.

Unfortunately, something goes wrong and the image segmenter is unloaded, leaving a hole in the computation graph and interrupting the flow of position information. This perturbation is detected as

$$\delta_p = (\varnothing, \{\langle cid_4, tid_4, [\texttt{/topic\_4}], \texttt{/topic\_3}\rangle\}, \varnothing, \varnothing, \varnothing, \varnothing). \tag{6.5}$$

The subsampler is still producing a stream of low-resolution images, but the segmenter no longer exists to do anything with them. The environment is now as in Figure 6.3 but without a segmenter. This perturbation results in the update procedure generating a change set by re-using the part of CU that still exists, but instantiating a new computation unit cid_8 of type tid_4 and reconfiguring it to subscribe to the streams that were already being produced by the subsampler, i.e.

$$\delta^* = (\{\langle cid_8, tid_4, [\texttt{/topic\_4}], \texttt{/topic\_3}\rangle\}, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing). \tag{6.6}$$

The detector's subscription to the defunct segmenter is thus replaced by one to the new segmenter, and the information flow is restored.

Some time later, Puff joins Piff on the field and registers its own transformations in accordance with Table 6.1, where piff is replaced by puff for new TIDs 8–14. The computational environment looks like before, but $|F_\varepsilon| = 14$ now. Given the possibility to generate a second pipeline for ball positions, the life-cycle daemon nevertheless does not use the second pipeline as-is. The reason for this is that the cost for re-using Piff's part of the computation graph is assumed to be free, whereas a lot of effort would have to be spent in order to instantiate Puff's pipeline to switch away from Piff's stream. Only if Puff's alternatives are significantly cheaper to make up for the extra labour cost will the daemon switch pipelines. Since both Piff and Puff are NAO robots with similar equipment and the same transformations, we assume this is not the case.

At some point, Piff needs to recharge its batteries. It therefore first deregisters its transformations from the environment, i.e.

$$\delta_p = (\varnothing, \varnothing, \varnothing, F^-, \varnothing, \varnothing), \tag{6.7}$$

where $|F^-| = 7$ consists of the seven transformations from Table 6.1. The life-cycle daemon correctly identifies the perturbation and starts a review interval, in which it determines that all active CUs are now defunct. This means that there is no longer any guarantee that the CUs provide streaming data. Thankfully, Puff's transformations provide a suitable replacement for the defunct CUs, leading to a change set

$$\delta^* = (CU^+, CU^-, \varnothing, \varnothing, \varnothing, \varnothing), \tag{6.8}$$

where the CU additions are based on Puff's transformations, and the CU removals are Piff's defunct CUs;

$$
\begin{aligned}
CU^{-} = \{\ & \langle cid_1, tid_7, [\texttt{/topic\_1}], \texttt{/target\_1} \rangle, \\
& \langle cid_2, tid_6, [\texttt{/topic\_2}, \texttt{/topic\_6}], \texttt{/topic\_1} \rangle, \\
& \langle cid_3, tid_5, [\texttt{/topic\_3}, \texttt{/topic\_4}], \texttt{/topic\_2} \rangle, \\
& \langle cid_8, tid_4, [\texttt{/topic\_4}], \texttt{/topic\_3} \rangle, \\
& \langle cid_5, tid_3, [\texttt{/topic\_5}], \texttt{/topic\_4} \rangle, \\
& \langle cid_6, tid_2, [], \texttt{/topic\_5} \rangle, \\
& \langle cid_7, tid_1, [], \texttt{/topic\_6} \rangle \},
\end{aligned} \tag{6.9}
$$

$$
\begin{aligned}
CU^{+} = \{\ & \langle cid_9, tid_{14}, [\texttt{/topic\_7}], \texttt{/target\_1} \rangle, \\
& \langle cid_{10}, tid_{13}, [\texttt{/topic\_8}, \texttt{/topic\_12}], \texttt{/topic\_1} \rangle, \\
& \langle cid_{11}, tid_{12}, [\texttt{/topic\_9}, \texttt{/topic\_10}], \texttt{/topic\_2} \rangle, \\
& \langle cid_{12}, tid_{11}, [\texttt{/topic\_10}], \texttt{/topic\_9} \rangle, \\
& \langle cid_{13}, tid_{10}, [\texttt{/topic\_11}], \texttt{/topic\_10} \rangle, \\
& \langle cid_{14}, tid_9, [], \texttt{/topic\_11} \rangle, \\
& \langle cid_{15}, tid_8, [], \texttt{/topic\_12} \rangle \}.
\end{aligned} \tag{6.10}
$$

Note also the removal of the cid_8 CU which was previously used to patch a gap in the computation graph.

After the occurence of the perturbation δ_p, the stream being produced on `/target_1` is temporarily interrupted as the review interval is performed. At the end of the review interval, the change set δ^* will have been applied to the environment, cancelling out the suboptimality imposed by δ_p by repairing the computation graph with an alternative pipeline satisfying the target. Consequently, the MSTL formula can be evaluated further. The system operates on a best-effort basis by quickly finding ways to repair broken computation graphs, thereby minimising the interruption in the streams used to construct a state stream for formula evaluation. Future work could look into ways to mitigate any data loss that may occur during these controlled hand-overs between agents.

6.4 Exploitation of new optima

Continuing the scenario, Puff is currently observing the ball which has not yet left the circle on the field, and Piff is in the process of recharging. The only transformations available to the computational environment are therefore Puff's. However, the lab itself is also equipped with four cameras. These cameras can be used in unison to generate a top-down image of the field. More importantly, these cameras could be used to locate objects on the field, in particular NAO robots and balls. A system can obtain the cam-

TID	TF label		Tags
tid_{15}	ceiling_cam(cam_1)		$\varnothing$
		$\Rightarrow$	rgbImageDistorted(cam_1)
tid_{16}	undistort(cam_1)		rgbImageDistorted(cam_1)
		$\Rightarrow$	rgbImage(cam_1)
tid_{17}	ceiling_cam(cam_2)		$\varnothing$
		$\Rightarrow$	rgbImageDistorted(cam_2)
tid_{18}	undistort(cam_2)		rgbImageDistorted(cam_2)
		$\Rightarrow$	rgbImage(cam_2)
tid_{19}	ceiling_cam(cam_3)		$\varnothing$
		$\Rightarrow$	rgbImageDistorted(cam_3)
tid_{20}	undistort(cam_3)		rgbImageDistorted(cam_3)
		$\Rightarrow$	rgbImage(cam_3)
tid_{21}	ceiling_cam(cam_4)		$\varnothing$
		$\Rightarrow$	rgbImageDistorted(cam_4)
tid_{22}	undistort(cam_4)		rgbImageDistorted(cam_4)
		$\Rightarrow$	rgbImage(cam_4)
tid_{23}	stitch(field)		rgbImage(cam_1)
			rgbImage(cam_2)
			rgbImage(cam_3)
			rgbImage(cam_4)
		$\Rightarrow$	rgbImage(field)
tid_{24}	ball_detector(field)		rgbImage(field)
		$\Rightarrow$	position(ball)

Table 6.2: The Humanoid lab's ceiling camera transformations and their tags denoted by **itag** $\Rightarrow$ *otag*.

era video feeds, stitch the images together, and perform localisation, without the computational limitations imposed by NAO hardware.

We can thus register the ceiling camera system with the computational environment, resulting in transformations with TIDs 15–24. The pipeline consists of the transformations shown in Table 6.2. The registration of these transformations constitutes a perturbation

$$\delta_p = (\varnothing, \varnothing, F^+, \varnothing, \varnothing, \varnothing), \tag{6.11}$$

where the set of added transformations $|F^+| = 10$ consists of the ten transformations listed in Table 6.2. Even though the perturbation did not break anything—there is still a stream on topic target_1—it is nevertheless a potential long-term positive perturbation, because the added transformations might yield cheaper-cost solutions. If this is not the case, then $\delta^* = \delta_\varnothing$; otherwise, the stream reasoning manager can replace part of the active pipeline with the transformations from the ceiling cameras. Since the perturbation is not a short-term negative perturbation, no immediate response is required, so the life-cycle daemon waits with the review cycle until the

horizon is reached or a short-term negative perturbation occurs.

Note that the ceiling camera pipeline provided by the lab lacks a transformation that can provide InsideCircle(ball) information. This is because that pipeline does not have the background knowledge to understand what InsideCircle means; it does not care about the lines on the field, but only about NAO robots and balls and their positions. However, Puff *does* care about the circle on the field, and therefore knows given a position whether that position is within the circle. Assuming that the reduction in upkeep outweighs the labour cost of switching pipelines, DyKnow-ROS finds a solution during the next review interval. It uses the entire ceiling camera pipeline plus the circle monitor from Puff, while unloading the remaining CUs. This leads to a change set

$$\delta^* = (CU^+, CU^-, \varnothing, \varnothing, \varnothing, \varnothing), \tag{6.12}$$

where the sets of CU additions and removals are described by

$$\begin{aligned} CU^- = \{ & \langle cid_{10}, tid_{13}, [\texttt{/topic\_8}, \texttt{/topic\_12}], \texttt{/topic\_1} \rangle, \\ & \langle cid_{11}, tid_{12}, [\texttt{/topic\_9}, \texttt{/topic\_10}], \texttt{/topic\_2} \rangle, \\ & \langle cid_{12}, tid_{11}, [\texttt{/topic\_10}], \texttt{/topic\_9} \rangle, \\ & \langle cid_{13}, tid_{10}, [\texttt{/topic\_11}], \texttt{/topic\_10} \rangle, \\ & \langle cid_{14}, tid_{9}, [], \texttt{/topic\_11} \rangle, \\ & \langle cid_{15}, tid_{8}, [], \texttt{/topic\_12} \rangle \}, \end{aligned} \tag{6.13}$$

$$\begin{aligned} CU^+ = \{ & \langle cid_{16}, tid_{24}, [\texttt{/topic\_13}], \texttt{/topic\_7} \rangle, \\ & \langle cid_{17}, tid_{23}, [\texttt{/topic\_14}, \texttt{/topic\_15}, \texttt{/topic\_16}, \\ & \quad \texttt{/topic\_17}], \texttt{/topic\_13} \rangle, \\ & \langle cid_{18}, tid_{22}, [\texttt{/topic\_18}], \texttt{/topic\_14} \rangle, \\ & \langle cid_{19}, tid_{20}, [\texttt{/topic\_19}], \texttt{/topic\_15} \rangle, \\ & \langle cid_{20}, tid_{18}, [\texttt{/topic\_20}], \texttt{/topic\_16} \rangle, \\ & \langle cid_{21}, tid_{16}, [\texttt{/topic\_21}], \texttt{/topic\_17} \rangle, \\ & \langle cid_{22}, tid_{21}, [], \texttt{/topic\_18} \rangle, \\ & \langle cid_{23}, tid_{19}, [], \texttt{/topic\_19} \rangle, \\ & \langle cid_{24}, tid_{17}, [], \texttt{/topic\_20} \rangle, \\ & \langle cid_{25}, tid_{15}, [], \texttt{/topic\_21} \rangle \}. \end{aligned} \tag{6.14}$$

Note that CU cid_9 is not among the CUs being unloaded as it is being reused in the new environment. The topic /topic_7 is therefore being reused as the output channel for the CU cid_{16}. The resulting environment has a cheaper long-term cost than would have been acrued by not performing the update, while still generating a stream on /target_1 with minimal interruption. When at some point Puff also requires recharging, it is possible for Puff to

leave while keeping the circle monitor available to the system. From that point on, the ceiling camera generate position data which is processed by Puff off-site. The system has successfully exploited the potential improvement to the configuration when it became available.

6.5 Cleaning up

Finally, with the field devoid of NAO robots and perhaps at the start of a new day, an unsuspecting student enters the lab and, not realising the experimental setup, takes the ball. The $\mathsf{InsideCircle}(\mathsf{ball})$ predicate evaluates to false, thereby violating the MSTL formula, yielding 'false' as its final answer. The stream reasoning manager releases the targets, corresponding to a perturbation

$$\delta_p = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \{\langle \mathsf{target1}, \mathsf{InsideCircle}(\mathsf{ball}), \texttt{/target\_1} \rangle\}). \tag{6.15}$$

With the target removed, the DyKnow-ROS life-cycle daemon immediately starts a review interval. Since no targets exist, any active CUs are needlessly expending upkeep cost. Therefore, all active CUs are removed while retaining the set of transformations;

$$\delta^* = (\varnothing, CU_\varepsilon, \varnothing, \varnothing, \varnothing, \varnothing). \tag{6.16}$$

Without any targets, the environment remains idle until new targets are registered, either as the result of a formula needing evaluation or because of another purpose for needing a semantic subscription.

6.6 Open problems

- Targets currently only consider cost, without considering *quality*. This prevents certain solutions from being chosen if they are more expensive, regardless of their quality being greater than that of cheaper solutions. As an example, sometimes redundant information can be useful. One situation wherein this is the case is sensor fusion. Given multiple sources of position information for an object, combining these sources may lead to a better position estimation. However, since this requires multiple pipelines and thus more upkeep costs, these solutions will never be chosen. It would be interesting to see how one could extend the approach presented here to a multi-target optimisation problem in which the cost is minimised and the quality is maximised.

- The synergy effect is demonstrated in terms of reasoning about streams supporting robust reasoning over streams in situations wherein the set of available computational resources changes. We have not yet

explored in detail the opposite synergy direction, wherein reasoning over streams may affect the reasoning about streams. This is a topic left for future work.

- The lack of multi-agent support at this stage means that the two NAO platforms used in this case study were part of a single DyKnow instance. Effectively it was the lab that acted as an agent. Separating the two platforms over two different DyKnow instances brings new challenges.

6.7 Summary

This chapter covered a case study wherein the DyKnow stream reasoning framework was deployed on NAO platforms. It illustrated how an instance of the stream reasoning framework proposed in this thesis can be used to effectively reason both over and about streams. The synergy effect was demonstrated by having reasoning about streams support the ability to robustly reason over streams, even if the set of available resources changes during the reasoning process.

Chapter 7

Related work

Work towards stream reasoning has resulted in many different perspectives and foci. This chapter serves as a survey of recent and/or ongoing research projects towards stream reasoning. The survey is not meant to be exhaustive, and does not include many of the well-known stream processing tools and libraries. The listed works are compared to the contributions presented in this thesis. Some of the listed work refers to specific projects; others represent areas of research of high relevance to this thesis.

7.1 LARS

LARS is a **L**ogic-based framework for **A**nalysing **R**easoning over **S**treams by Beck et al. (2014, 2015) and provides a logical formalisation of stream reasoning. LARS considers stream reasoning to be logical reasoning on streaming data, and therefore takes an approach wherein streaming data is modelled logically, i.e. as predicates. This approach shares similarities with DyKnow's state streams, which carry the truth values of predicates over time as well. Unlike DyKnow, however, LARS does not consider the production of state streams.

Key contributions presented as part of the LARS framework are reported (Beck et al., 2015) to include

1. a rule-based formalism for reasoning over streams;
2. different means to refer to or abstract from time; and
3. a window operator to this effect.

The window operator $\boxplus^{\mathbf{x}}_{\iota,ch}$ is applied to a stream S in order to produce a resulting stream S', where ι indicates a window type, ch a stream choice function, and $\mathbf{x}$ a vector of window parameters. The window type ι is used to identify a window function w_ι. It maps from an input stream S, a reference (starting) time point t, and parameters $\mathbf{x}$ to a *substream* $S' \subseteq S$.

LARS has successfully modelled time-based, tuple-based and partition-based windows, making it expressive enough to capture languages such as CQL (Arasu et al., 2003, 2006).

LARS' window operator can be used to filter elements from a stream and apply logical reasoning to the resulting substream, thereby providing different potential views. In the DyKnow model, a window operator would instead exist as a transformation that filters a stream based on windowing conditions, rather than be part of the logical representation. DyKnow's computational environment can also make a distinction between a filtering operation akin to the LARS substream-producing windowing operation on the one hand, and the case wherein every sample contains a window on the other hand. It is presently unclear how this distinction could be leveraged in the LARS framework. In conclusion, LARS shares similarities with DyKnow in terms of reasoning with the help of transformations on streams, which allow LARS to switch views and make logical statements on those views.

7.2 SECRET

Similar to LARS, *SECRET* is a model for analysing the execution semantics of stream processing systems proposed by Botan et al. (2010). The motivation behind SECRET is rooted in the existence of multiple stream processing engines, each with their own capabilities and semantics, and the desire to compare the execution behaviour of these heterogeneous stream processing engines. In particular, the heterogeneity manifests itself in terms of syntax, capability, and the execution model. SECRET is (arguably loosely) named after the four dimensions it considers; **s**cop**e**, **c**ontent, **re**port and **t**ick. Dindar et al. (2013) consider these four dimensions with SECRET in their coverage of the heterogeneity of the Coral8, STREAM, StreamBase, and Oracle CEP stream processing engines.

SECRET considers streams to be countably infinite sets of elements $s \in \mathbb{S}$, such that a stream element (or a sample in DyKnow's terminology) is described by $\langle v, t^{app}, t^{sys}, tid, bid \rangle$. Here v denotes a relational tuple conforming to a schema S (i.e. a table), $t^{app}, t^{sys} \in \mathbb{T}$ denote the application time and system time, and tid, bid denote tuple ID and batch ID values. This type formalisation of a stream is similar to DyKnow, which considers named structured values that could be represented as done in SECRET. A batch $\mathbb{B}$ is described as a set of stream elements such that each element making up a batch has the same t^{app} as all other elements of that same batch. State streams in DyKnow could thus be described in terms of batches. Finally, as in LARS, SECRET describes a variety of window semantics using the definition of a stream, where a window over a stream produces a substream. In particular, SECRET describes time-based windows and tuple-based windows with varying window sizes and slides.

A key motivation for SECRET was the heterogeneity in the window operations supported by various stream processing engines. SECRET thus captures the window-based query execution semantics along the aforementioned four dimensions. **Scope** deals with the scope of a window, meaning the window intervals, given a set of parameters. Scope can be interpreted differently by different stream processing systems. **Content** deals with how the scope of these windows translates into the content of the produced substreams given an input stream. The content is then commonly sent on for processing, such as for example aggregation. *When* the content becomes visible to the query processor can vary by system. **Report** states the conditions on when content becomes visible. Lastly, **tick** deals with the control loop of a stream processing engine, and in particular *when* it acts on a given input stream. Given these four dimensions, Dindar et al. (2013) consider both time-based and tuple-based windows for the aforementioned stream processing engines.

SECRET is primarily a tool for analysing different stream processing engines. As with LARS, SECRET has some overlap with the formal specifications of DyKnow. The main difference between LARS and SECRET appears to be the level of detail; LARS provides high-level semantics relative to a logical model, whereas SECRET is closer to the operational semantics of a set of pre-existing stream processing engines. In both approaches, the semantic of the window operator were a primary point of attention. DyKnow currently does not support window operations directly, although windowing does take place in the form of interval-bounded temporal operators. Nevertheless, SECRET's formal specification of window operations can be of use when considering similar operations such as merging and synchronisation as part of for example state stream generation in DyKnow.

7.3 RSP

RDF Stream Processing (RSP) refers to stream processing techniques that assume streaming data to be formatted in the RDF data format. Many querying languages have been designed for RSP, usually based on a continuous version of the SPARQL query language for RDF graphs. These languages include but are not limited to C-SPARQL (Barbieri et al., 2009), CQELS (Le-Phuoc et al., 2011), SPARQL$_{stream}$ (Calbimonte et al., 2010) and EP-SPARQL (Anicic et al., 2011).

RSP originally continued the same pattern forming the basis for efforts such as LARS or SECRET; different RSP implementations used different semantics for windowing operations, resulting in different answers depending on the system used. While the representation of RDF graphs is well-defined, the content of RDF streams is not. Furthermore, since operations on RDF graphs were time-invariant (incorporating time into ontologies is a difficult open problem), combining streams with ontologies resulted in different approaches. RSP-QL was therefore proposed by (Dell'Aglio et al.,

2014) as a unifying query model to explain the heterogeneity of these various RSP languages. To this effect, it extends the SPARQL model and bases off the CQL and SECRET models.

7.4 PEIS

Research towards analysis of stream reasoning such as proposed as part of LARS, SECRET and to some extent RSP generally ignores questions of integration into a larger (eco)system. Saffiotti et al. (2008) presented the *PEIS ecology*[13] for **P**hysically **E**mbedded **I**ntelligent **S**ystems. The cornerstone of the PEIS ecology is its conceptualisation of physically embedded intelligent systems (PEIS) as agents that operate in a physical environment and are themselves physical entities. Every PEIS is assumed to at least have

1. some computational resources;
2. some communication resources; and
3. sensors and/or actuators allowing the system to interact with the physical environment.

Consequently, PEIS are assumed to be heterogeneous entities with different capabilities. A PEIS ecology consists of potentially many PEIS, each with their own functionalities and communication capabilities. While the PEIS ecology considers communication problems, DyKnow instead chooses to use ROS as a commonly-used platform that provides communication support. The PEIS ecology as a whole is intended to solve problems in a multi-agent organisation setting by interacting with the physical environment.

Lundh et al. (2008) focused on the problem of self-configuration and proposed techniques for configuration planning. The underlying motivation is that in the PEIS ecology robots can and should help other robots to collectively achieve goals common to the ecology they are part of. Functionalities are formalised in a logical representation that can be used by general planners. Given a goal, the planner is able to find a set of functionalities that, when activated, fulfill the goal. This approach shares similarities with DyKnow's semantic subscriptions. Both consider a computational environment in which functionalities can be activated or transformations can be instantiated for a cost. However, in DyKnow this cost is estimated and may change over time, whereas the PEIS ecology uses simple constant values. Furthermore, DyKnow's similarity relation is based on the semantic tags of transformations, whereas the PEIS ecology matches propositional statements. Both the lack of meaningful cost measures and the potential value in using semantic descriptions were later identified (Lundh, 2009) as future work. On the other hand, the PEIS ecology is able to model

[13] Pronounced '*pace* ecology'

actions taken by PEIS at the level of configuration planning, whereas DyKnow can only consider stream processing without taking into account the actions of agents. The preconditions for transformations are not explicitly modeled in DyKnow either; transformations are expected to only be available when preconditions are met, as exemplified in the synergy scenarios. DyKnow focuses to a large degree on maintaining semantic subscriptions and therefore emphasises the need for efficient and fast reconfiguration in light of failures. The PEIS ecology instead focuses on achieving a goal in a physical environment, where the configuration of functionalities of PEIS plays one role. DyKnow and the PEIS ecology are thus complementary in their results, where the difference in motivations means there is a different focus.

Moving from the configuration-centric abstraction level down to the data-centric abstraction level, Alirezaie (2015) more recently focused on the problem of streaming data semantics. In particular, the focus was on bridging the semantic gap between sensor data and ontological knowledge, which is reminiscent of the sense-reasoning gap that was the motivation (Heintz et al., 2010) behind earlier DyKnow efforts. The semantic gap between sensor data and ontological knowledge is described as the disconnect between quantitative sensor values and crisp high-level knowledge encoded into ontologies. Alirezaie (2015) focuses on two aspects. First, correspondences between sensor data and conceptual knowledge needs to be automatically determined. Second, the two types of information are combined in an inferencing process. In particular, the focus is on enriching the sensor data, meaning it is 'lifted up' to the conceptual level. This is different from DyKnow's approach of describing the low-level sensor information using high-level concepts, as this is purely descriptive rather than formative. The use of CEP on semantic events obtained from sensor information is an interesting approach currently not used by DyKnow.

Overall, the PEIS ecology shares many similarities with the DyKnow project. Both efforts consider a larger integration problem in which stream reasoning combining sensor data with high-level knowledge is essential for decision-making, albeit from different angles.

Chapter 8

Conclusions and future work

In this thesis we presented a logic, algorithms, formal models, semantic representations, integration, a concrete implementation, and a case study for spatio-temporal stream reasoning with adaptive state stream generation. The logic MSTL was used to make spatio-temporal statements of which the truth value can be robustly determined even in the face of unexpected changes in the availability of (latent) streams. The thesis is quite broad in its scope, resulting in the focus on the development and integration of two related strands. In this chapter, we take a critical look at the resulting contributions. We therefore cover not just the intentional limitations, but also unintentional limitations when considering the results from other areas within computer science. Finally, we consider the potential for future works, and provide a discussion of these options.

8.1 Conclusions and lessons learned

The results presented in this thesis represent the latest achievements within the DyKnow project, divided into two integrated strands. The spatio-temporal stream reasoning results form the first strand. Here, MSTL is presented as an extension of MTL by incorporating RCC-8 for qualitative spatial reasoning, allowing for spatio-temporal statements to be made. The truth value of these statements can be determined incrementally using an extended version of progression. These statements can further contain intertemporal spatial relations similar to ST_1. Importantly, we assume that these intertemporal spatial relations cannot be observed directly, and thus need to be inferred. Without any additional information about intertemporal relations, nothing is known about them. Our solution therefore makes use of landmark regions which can reduce the uncertainty over intertemporal

spatial relations.

The second strand focuses on the problem of generating a state stream over which an MSTL formula can be evaluated. The symbols in a formula are therefore grounded in a computational environment, such that the truth value of these symbols depends on the data that is produced by this underlying environment. Semantic annotations of the logical symbols (through the use of targets) as well as the available stream transformations allows us to find suitable configurations of the computational environment that produce a state stream containing the information necessary to evaluate a formula. By reconsidering the configuration periodically, the computation graphs can be repaired or improved in case where the underlying system changes unexpectedly. This ensures that the progression of an MSTL formula is not necessarily interrupted or fails as the result of such changes, making the system more robust. Additionally, the configurations can be expressed relative to a Semantic Web ontology, allowing for the exchange of configuration information.

The two strands were integrated into a single stream reasoning framework in which reasoning about streams synergises with reasoning over streams. The resulting framework model was integrated with ROS and allows existing ROS nodelet implementations to be used in DyKnow with minimal overhead in terms of throughput and developer burden. This concrete implementation was then deployed on NAO platforms, adapting software produced by the Linköping RoboCup SPL team to be usable by DyKnow for a case study that highlights the added value of adaptive reconfigurability during stream reasoning tasks.

While the focus of the work was primarily on robotic applications, the solutions are general and do not rely on specific supporting software such as ROS. For example, experimental CUs have been written for non-robotic domains such as Twitter, or to interact with DigitalOcean's API by instantiating, managing, and destroying virtual machines in off-platform data centres. This highlights potential applicability of the presented solutions to much broader application areas that involve many diverse computational resources, for example smart cities or sensor networks, making them potentially interesting to industrial applications of this kind. The computation resources also do not necessarily have to be physical. One can imagine virtual services that deal with areas such as advertisement, travel agencies, or stock markets wherein financial information and their sources may change continually. In fact, many CEP languages have query examples that deal precisely with stock market events.

8.2 Limitations and open problems

We already discussed chapter-specific limitations and open problems throughout the thesis, but a global view has not yet been covered. The work pre-

sented in this thesis has a number of limitations by design, which were initially discussed in the introduction.

We focused only on logic-based stream reasoning. This was a clear choice made from the beginning to see what we could do with temporal logics once spatial information gets involved. The solutions presented were designed with robot applications in mind, but as pointed out before, they need not be exclusive to robot applications. Nevertheless, it is one of the reasons why we could not assume that our data was in the RDF data format, as is done by many Semantic Web applications towards stream reasoning. From an architectural point of view, there are some obvious limitations as well. The synergy effect is primarily one-sided in the presentation of this thesis. Reasoning about streams does indeed support reasoning over streams by producing necessary state streams. Likewise, reasoning over streams does influence reasoning about streams by resulting in the generation of targets. However, more work could be done in the latter case by for example considering the addition and removal of transformations based on the truth value of formulas.

Taking a step back from the theoretical and practical contributions, one can also consider the motivation for the work. Practically, today, many of the problems presented as example scenarios in this thesis could be solved more easily *ad-hoc*. However, the base assumption of research into for example the IoT is that there will be many *things*, requiring a more scalable approach. Smart cities have been mentioned before as a potential application area, yet no experiments were performed at that scale. Our Humanoid lab case study does not convey the scalability issue in the same way as a smart city would. This is an example of an unintentional limitation of the work.

8.3 Future work

While the aforementioned open problems are an obvious target for future work, they are quite specific. More long-term future work could focus on a number of areas, and in particular areas that are relevant but have not been the focus of this work. For the stream reasoning, it would be interesting to directly work with incomplete information. The usage of RCC-8 led to situations wherein there is uncertainty in the information inferred. This is an issue that could also apply to other calculi such as Allen's interval algebra. Having a way to reason specifically about the incompleteness of information could allow one to make logical statements that deal with the incompleteness explicitly. Additionally, the support of probabilistic reasoning would be extremely useful in robotic scenarios, as in many cases the information we want to use in the crisp logical formulas is actually represented in terms of probability distributions. While it is trivial to provide mean values, this does not handle Boolean comparisons nicely, as a distribution

might overlap with a threshold, thus making the truth value of the comparison inherently probabilistic. This also impacts the way state streams are generated, as more meta-information is required to properly combine probabilistic information of this kind. One interesting use-case would be that of automated fusion, wherein the underlying configuration manager takes into account the possibility of fusing probabilistic data streams in certain contexts.

There remains a lot of potential future work in the adaptive state stream generation strand, in addition to the limitations mentioned earlier. In particular, determining appropriate utility measures with meaningful properties is an issue. For example, if we can provide a higher-quality stream by fusing two probabilistic streams, there is still a trade-off to be made in terms of the labour and upkeep such a reconfiguration would cost. Finding a suitable trade-off between cost and utility is an important problem especially for robot applications.

Furthermore, the current solution is designed with a single agent in mind. By expanding reasoning over and about streams to a multi-agent system setting, we can consider many interesting problems in addition to the ones described above. While there exists ongoing work into configuration of for example cloud computing systems, these approaches commonly have data centres in mind. Extending these techniques and others to heterogeneous autonomous robot applications would be interesting.

Finally, further investigation of the synergy effect resulting from reasoning about and over streams may be of interest to many problems not limited to situation awareness. Being able to reason about one's own percepts allows one to potentially resolve inconsistencies. As a motivating example, consider an agent exploring a new continent and stumbling upon what appear to be black swans. This finding is revolutionary yet troublesome because, as far as the agent is aware, all swans are white—a Black Swan event. How can we resolve this inconsistency? One way is the adoption of well-studied non-monotonic defeasible reasoning techniques that allow for useful rules with exceptions or belief updates; swans are usually white, but black swans exist.[14] This type of resolution occurs in the area of 'reasoning over streams'. A different resolution can be found when we consider reasoning about streams. Clearly our agent's percepts led us to deduce an inconsistency given our belief base is assumed to be correct. Therefore, the agent can attempt to verify its observations over time by attempting to either corroborate or contradict the observations, using other modes of perception. The underlying consideration is close to that of Plato's cave; the agent is observing shadows created by the outside worlds and may be fooled by those shadows. Perhaps the camera is faulty, or the means of detecting swans is yielding incorrect detections. Logically either the observations are wrong, or the reasoning is. By reasoning about streams, an

[14] This is of course a variant of the 'Tweety' scenario wherein it is assumed that all birds can fly, but that Tweety is a penguin, and penguins cannot fly.

agent is able to reason about perception itself and could thus find alternate modes of perception to either corroborate the contradiction or contradict the inconsistent observation. The value of such corroborating or contradictory evidence is of course subject to the philosophy of science. This thesis presents but a few initial steps towards such an agent from the starting point of stream reasoning.

Appendix A

DyKnow ontology in Manchester syntax

The following is a listing of the DyKnow ontology used for semantic interoperability. It makes use of Manchester syntax to improve human readability. The full up-to-date ontology in OWL/RDF syntax can be obtained from `http://www.dyknow.eu/ontology/` in accordance with the dereferencing guidelines for ontologies. We use Parrot for HTML requests.

```
Prefix: : <http://www.dyknow.eu/ontology/dyknow#>
Prefix: dc: <http://purl.org/dc/elements/1.1/>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: skos: <http://www.w3.org/2004/02/skos/core#>
Prefix: terms: <http://purl.org/dc/terms/>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>



Ontology: <http://www.dyknow.eu/ontology/dyknow>
<http://www.dyknow.eu/ontology/dyknow/201707>

Annotations:
    terms:creator "Daniel de Leng"^^xsd:string,
    terms:modified "2017-07-27",
    rdfs:comment "The DyKnow ontology can be used as a common
         representation of stream reasoning framework configurations."@en,
    rdfs:label "DyKnow Ontology"@en

AnnotationProperty: rdfs:comment


AnnotationProperty: rdfs:label


AnnotationProperty: terms:creator

```

```
30
31  AnnotationProperty: terms:modified
32
33
34  Datatype: rdf:PlainLiteral
35
36
37  Datatype: xsd:Name
38
39
40  Datatype: xsd:anyURI
41
42
43  Datatype: xsd:date
44
45
46  Datatype: xsd:dateTimeStamp
47
48
49  Datatype: xsd:string
50
51
52  ObjectProperty: dependsOn
53
54      SubPropertyChain:
55          hasSubscription o fromCU
56
57      Characteristics:
58          Transitive
59
60
61  ObjectProperty: fromCU
62
63      DisjointWith:
64          toCU
65
66      Characteristics:
67          Functional
68
69      Domain:
70          Subscription
71
72
73  ObjectProperty: fromPort
74
75      Characteristics:
76          Functional
77
78      Domain:
79          Subscription
80
81      Range:
82          OutPort
83
84
85  ObjectProperty: hasChannel
86
87      Characteristics:
88          Functional
89
```

```
    Domain:
        Subscription or Target

    Range:
        Channel


ObjectProperty: hasCostModel

    Characteristics:
        Functional


ObjectProperty: hasEnvironment

    Range:
        Environment


ObjectProperty: hasInPort

    Domain:
        Transformation

    Range:
        InPort

    InverseOf:
        isInPort


ObjectProperty: hasInstance

    Domain:
        Transformation

    InverseOf:
        instanceOf


ObjectProperty: hasOutPort

    Domain:
        Transformation

    Range:
        OutPort

    InverseOf:
        isOutPort


ObjectProperty: hasSample

    Characteristics:
        Functional

    Domain:
        SampleSequence

```

```
    Range:
        Sample


ObjectProperty: hasSampleSequence

    Characteristics:
        Functional

    Domain:
        Stream

    Range:
        SampleSequence


ObjectProperty: hasState

    Characteristics:
        Functional

    Domain:
        StateSequence

    Range:
        State


ObjectProperty: hasStateSequence

    Characteristics:
        Functional

    Range:
        StateSequence


ObjectProperty: hasSubscription

    Range:
        Subscription

    InverseOf:
        toCU


ObjectProperty: hasTag

    Range:
        Tag


ObjectProperty: hasTagDescription

    Characteristics:
        Functional

    Range:
        Tag

```

```
210
211 ObjectProperty: instanceOf
212
213     Range:
214         Transformation
215
216     InverseOf:
217         hasInstance
218
219
220 ObjectProperty: isInPort
221
222     Domain:
223         InPort
224
225     Range:
226         Transformation
227
228     InverseOf:
229         hasInPort
230
231
232 ObjectProperty: isOutPort
233
234     Domain:
235         OutPort
236
237     Range:
238         Transformation
239
240     InverseOf:
241         hasOutPort
242
243
244 ObjectProperty: nextSample
245
246     Characteristics:
247         Functional,
248         Irreflexive
249
250     Domain:
251         Sample
252
253     Range:
254         Sample
255
256
257 ObjectProperty: nextState
258
259     Characteristics:
260         Functional,
261         Irreflexive
262
263     Domain:
264         State
265
266     Range:
267         State
268
269
```

```
ObjectProperty: toCU

    DisjointWith:
        fromCU

    Characteristics:
        Functional

    Domain:
        Subscription

    InverseOf:
        hasSubscription


ObjectProperty: toPort

    Characteristics:
        Functional

    Domain:
        Subscription

    Range:
        InPort


DataProperty: hasChannelName

    Characteristics:
        Functional

    Domain:
        Channel

    SubPropertyOf:
        hasName


DataProperty: hasLabel

    Characteristics:
        Functional

    Range:
        xsd:Name


DataProperty: hasName

    Characteristics:
        Functional


DataProperty: hasPortName

    SubPropertyOf:
        hasName


```

```
330 DataProperty: hasTimeStamp
331
332     Characteristics:
333         Functional
334
335     Range:
336         xsd:dateTimeStamp
337
338
339 DataProperty: hasValue
340
341     Characteristics:
342         Functional
343
344
345 Class: ChangeSet
346
347     Annotations:
348         rdfs:comment "A change set describes changes made to an
                environment. Formally the change set at least describes the
                additions and removals of computation units, transformations,
                and targets."@en,
349         rdfs:label "Change Set"@en
350
351
352 Class: Channel
353
354     Annotations:
355         rdfs:label "Channel"@en,
356         rdfs:comment "Channels are named transportation mechanisms for
                data."@en
357
358     SubClassOf:
359         hasChannelName some xsd:string
360
361
362 Class: CostModel
363
364     Annotations:
365         rdfs:label "Cost Model"@en,
366         rdfs:comment "A model describing how to calculate the cost of an
                update."@en
367
368
369 Class: Environment
370
371     Annotations:
372         rdfs:label "Environment"@en,
373         rdfs:comment "An environment is composed of a set of computation
                units (sometimes called a computation graph), a set of
                transformations, a set of targets, and a similarity relation
                between tags. The environment can be changed by applying a
                change set to it. This application is called an update.
                Environments describe the state of a stream reasoning
                framework."@en
374
375     SubClassOf:
376         hasName some xsd:Name
377
378
```

```
Class: InPort

    Annotations:
        rdfs:label "Input Port"@en,
        rdfs:comment "A port for receiving streaming data over a channel."
            @en

    SubClassOf:
        Port

    DisjointWith:
        OutPort


Class: LabourCostModel

    Annotations:
        rdfs:label "Labour Cost Model"@en,
        rdfs:comment "A cost model for calculating the labour cost."@en,
        rdfs:label "Labor Cost Model"@en,
        rdfs:comment "A cost model for calculating the labor cost."@en

    SubClassOf:
        CostModel


Class: OutPort

    Annotations:
        rdfs:label "Output Port"@en,
        rdfs:comment "A port for transmitting streaming data over a
            channel."@en

    SubClassOf:
        Port

    DisjointWith:
        InPort


Class: Parameter

    Annotations:
        rdfs:label "Parameter"@en

    SubClassOf:
        hasLabel some xsd:Name,
        hasValue some xsd:anyURI


Class: Port

    Annotations:
        rdfs:comment "The connection between a channel and a computation
            unit is realised in terms of ports. Ports are named entities."
            @en,
        rdfs:label "Port"@en

    SubClassOf:
        hasPortName some xsd:Name
```

```
435
436
437 Class: Sample
438
439     Annotations:
440         rdfs:label "Sample"@en,
441         rdfs:comment "An atomic, time-stamped data point."@en
442
443     SubClassOf:
444         hasLabel some xsd:Name,
445         hasTimeStamp some xsd:dateTimeStamp,
446         hasValue some xsd:anyURI
447
448
449 Class: SampleSequence
450
451     Annotations:
452         rdfs:label "Sample Sequence"@en
453
454     EquivalentTo:
455         hasSample some Sample
456
457
458 Class: Sink
459
460     Annotations:
461         rdfs:comment "A transformation that does not produce any resulting
                stream is called a sink."@en,
462         rdfs:label "Sink"@en
463
464     SubClassOf:
465         Transformation,
466         hasOutPort exactly 0 OutPort
467
468
469 Class: Source
470
471     Annotations:
472         rdfs:comment "A transformation that does not take any incoming
              stream is called a source."@en,
473         rdfs:label "Source"@en
474
475     SubClassOf:
476         Transformation,
477         hasInPort exactly 0 InPort
478
479
480 Class: State
481
482     Annotations:
483         rdfs:comment "A state is a mapping from a variable to a value."@en
              ,
484         rdfs:label "State"@en
485
486     SubClassOf:
487         hasLabel some xsd:Name,
488         hasValue some xsd:anyURI
489
490
491 Class: StateSequence
```

```
492
493     Annotations:
494         rdfs:label "State Sequence"@en
495
496     EquivalentTo:
497         hasState some State
498
499
500 Class: StateStream
501
502     Annotations:
503         rdfs:label "State Stream"@en,
504         rdfs:comment "A stream composed of states is called a state stream
                . State streams thus describe mappings from sets of variables
                to sets of values for specific time-points. State streams can
                be used to for example evaluate logical formulas."@en
505
506     SubClassOf:
507         Stream
508
509
510 Class: Stream
511
512     Annotations:
513         rdfs:comment "A sequence of samples representing a flow of data is
                 called a stream."@en,
514         rdfs:label "Stream"@en
515
516     EquivalentTo:
517         hasSampleSequence some SampleSequence
518
519
520 Class: Subscription
521
522     Annotations:
523         rdfs:comment "A subscription is a connection from a transmitting
                port to a receiving port over a channel."@en,
524         rdfs:label "Subscription"@en
525
526     SubClassOf:
527         fromPort some Port,
528         hasChannel some Channel,
529         toPort some Port
530
531
532 Class: Tag
533
534     Annotations:
535         rdfs:label "Tag"@en,
536         rdfs:comment "A tag is a descriptor with which concepts can be
                annotated. A concrete application can extend the Tag concept
                to describe an annotation language."@en
537
538     SubClassOf:
539         hasTagDescription some owl:Thing
540
541
542 Class: Target
543
544     Annotations:
```

```
        rdfs:comment "Targets describe the semantics of a desired
             information stream by using tags. Every target specifies a
             channel over which this desired information should be sent.
             Targets can be used to obtain adaptive semantic subscriptions
             which can be maintained by a DyKnow stream reasoning manager."
             @en,
        rdfs:label "Target"@en

    SubClassOf:
        hasChannel some Channel,
        hasTag some Tag,
        hasName some xsd:Name


Class: Transformation

    Annotations:
        rdfs:comment "Transformations describe stream-generating functions
             over streams that can be instantiated as computation unit.
             The act of instantiating a transformation results in cost
             being acrued. Transformations are identifiable by a unique
             name."@en,
        rdfs:label "Transformation"@en

    SubClassOf:
        hasCostModel some LabourCostModel,
        hasName some xsd:Name


Class: UpkeepCostModel

    Annotations:
        rdfs:comment "A cost model for calculating the upkeep cost."@en,
        rdfs:label "Upkeep Cost Model"@en

    SubClassOf:
        CostModel


Class: owl:Thing
```

Bibliography

M. Alirezaie. *Bridging the Semantic Gap between Sensor Data and Ontological Knowledge*. PhD thesis, Örebro university, 2015.

J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *Proceedings of the 20th International World Wide Web Conference (WWW)*, 2011.

A. Arasu, S. Babu, and J. Widom. Cql: A language for continuous queries over streams and relations. In *Proceedings of the 9th International Workshop on Database Programming Languages (DBPL)*, pages 1–19. Springer, 2003.

A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. *STREAM: The Stanford data stream management system*, pages 317–336. Stanford InfoLab, 2004.

A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proceedings of the 13th AAAI conference of Artificial Intelligence*, pages 1215–1222, 1996.

F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27, 1998.

D. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International World Wide Web Conference (WWW)*, 2009.

H. Beck, M. Dao-Tran, T. Eite, and M. Fink. Towards a logic-based framework for analyzing stream reasoning. In *Proceedings of the 3rd International Workshop on Ordering and Reasoning (OrdRing)*, 2014.

H. Beck, M. Dao-Tran, T. Eiter, and M. Fink. LARS: A logic-based framework for analyzing reasoning over streams. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

B. Bennett, A. Cohn, F. Wolter, and M. Zakharyaschev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, 2002.

T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. J. Miller, and N. Tatbul. SECRET: a model for analysis of the execution semantics of stream processing systems. *Proceedings of the VLDB Endowment*, 3(1-2):232–243, 2010.

A. Bröring, K. Janowicz, C. Stasch, and W. Kuhn. Semantic challenges for sensor plug and play. In *Proceedings of the International Symposium on Web and Wireless Geographical Information Systems (W2GIS)*, volume 5886, pages 72–86, 2009.

A. Bröring, P. Maué, K. Janowicz, D. Nüst, and C. Malewski. Semantically-enabled sensor plug & play for the sensor web. *Sensors*, 11(8):7568–7605, 2011.

J.-P. Calbimonte, O. Corcho, and A. J. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, pages 96–111. Springer, 2010.

A. Cohn and J. Renz. Qualitative spatial representation and reasoning. In *Handbook of Knowledge Representation*, pages 869–886. Elsevier, 2008.

M. Compton et al. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32, 2012.

G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

Z. Cui, A. G. Cohn, and D. A. Randell. Qualitative and topological relationships in spatial databases. In *Proceedings of the Third International Symposium on Advances in Spatial Databases (SSD)*, pages 296–315, 1993.

D. Dell'Aglio, E. Della Valle, J.-P. Calbimonte, and O. Corcho. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *International Journal on Semantic Web and Information Systems*, 10(4):17–44, 2014.

Y. Diao, N. Immerman, and D. Gyllstrom. SASE+: An agile language for Kleene closure over event streams, 2007.

N. Dindar, N. Tatbul, R. J. Miller, L. M. Haas, and I. Botan. Modeling the execution semantics of stream processing engines with SECRET. *The VLDB Journal*, 22(4):421–446, 2013.

P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 19(3):332–377, 2009.

Z. Dragisic. Semantic matching for stream reasoning. Master's thesis, Linköping University, 2011.

S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

Z. Gantner, M. Westphal, and S. Wölfl. GQR – a fast reasoner for binary qualitative constraint calculi. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 24–29, 2008.

A. Gerevini and B. Nebel. Qualitative spatio-temporal reasoning with RCC-8 and Allen's interval calculus: Computational complexity. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, volume 2, pages 312–316, 2002.

M. Ghallab. On chronicles: Representation, on-line recognition and learning. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, pages 597–606, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: Complex event processing over streams. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR)*, 2006.

F. Heintz. *DyKnow : A Stream-Based Knowledge Processing Middleware Framework*. PhD thesis, Linköping University, 2009.

F. Heintz and P. Doherty. DyKnow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 15(1), 2004.

F. Heintz and Z. Dragisic. Semantic information integration for stream reasoning. In *Proceedings of the 15th International Conference on Information Fusion (FUSION)*, 2012.

F. Heintz and D. de Leng. Semantic information integration with transformations for stream reasoning. In *Proceedings of the 16th International Conference on Information Fusion (FUSION)*, pages 445–452, 2013.

F. Heintz and D. de Leng. Spatio-temporal stream reasoning with incomplete spatial information. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 429–434, 2014.

F. Heintz, J. Kvarnström, and P. Doherty. Bridging the sense-reasoning gap: DyKnow – stream-based middleware for knowledge processing. *Journal of Advanced Engineering Informatics*, 24(1):14–26, 2010.

A. Hongslo. Stream processing in the Robot Operating System framework. Master's thesis, Linköping University, 2012.

J. Huang. Compactness and its implications for qualitative spatial and temporal reasoning. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012.

F. Kerasiotis, C. Koulamas, C. Antonopoulos, and G. Papadopoulos. Middleware approaches for wireless sensor networks based on current trends. In *Proceedings of the 4th Mediterranean Conference on Embedded Computing (MECO)*, pages 244–249, 2015.

R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev. Spatial logic + temporal logic = ? In *Handbook of Spatial Logics*, pages 497–564. Springer, 2007.

R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

D. Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber. A vision of swarmlets. *IEEE Internet Computing*, 19(2):20–28, 2015.

D. Lazarovski. Extending the stream reasoning in DyKnow with spatial reasoning in RCC-8. Master's thesis, Linköping University, 2012.

D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th International Conference on The Semantic Web*, ISWC'11, pages 370–388, 2011.

D. de Leng. Extending semantic matching in DyKnow to handle indirectly-available streams. Master's thesis, Utrecht University, 2013.

D. de Leng. Querying flying robots and other Things: Ontology-supported stream reasoning. *XRDS*, 22(2):44–47, 2015.

D. de Leng and F. Heintz. Towards on-demand semantic event processing for stream reasoning. In *Proceedings of the 17th International Conference on Information Fusion (FUSION)*, pages 1–8, 2014.

D. de Leng and F. Heintz. Ontology-based introspection in support of stream reasoning. In *Proceedings of the 1st Joint Ontology Workshops (JOWO) co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015a.

D. de Leng and F. Heintz. Ontology-based introspection in support of stream reasoning. In *Proceedings of the 13th Scandinavian Conference on Artificial Intelligence (SCAI)*, pages 78–87, 2015b.

D. de Leng and F. Heintz. Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, pages 957–963, 2016a.

D. de Leng and F. Heintz. DyKnow: A dynamically reconfigurable stream reasoning framework as an extension to the robot operating system. In *Proceedings of the 5th IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 957–963, 2016b.

D. de Leng and F. Heintz. Towards adaptive semantic subscriptions for stream reasoning in the robot operating system. In *Proceedings of the 30th IEEE/RSJ International Conference on Intelligent Robots and Systems (SIMPAR) (to appear)*, 2017.

J. J. Li, T. Kowalski, J. Renz, and S. Li. Combining binary constraint networks in qualitative reasoning. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, volume 8, pages 515–519, 2008.

R. Lundh. *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. PhD thesis, Örebro University, 2009.

R. Lundh, L. Karlsson, and A. Saffiotti. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems*, 56(10): 819–830, 2008.

C. Lutz and M. Miličić. A tableau algorithm for description logics with concrete domains and general TBoxes. *Journal of Automated Reasoning*, 38 (1-3):227–259, 2007. ISSN 0168-7433.

A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

D. Martin et al. OWL-S: Semantic markup for web services. *W3C member submission*, 2004.

D. L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 2004.

J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *Proceedings of the International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13, 2008.

E. Pejman, Y. Rastegari, P. M. Esfahani, and A. Salajegheh. Web service composition methods: A survey. In *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS)*, volume 1, pages 560–564, 2012.

J. M. T. Portocarrero, F. C. Delicato, P. F. Pires, T. C. Rodrigues, and T. V. Batista. SAMSON: Self-adaptive middleware for wireless sensor networks. In *Proceedings of the 31st ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 1315–1322, 2016.

M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source robot operating system. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 165–176, 1992.

J. Rao and X. Su. A survey of automated web service composition methods. In *Proceedings of the International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, volume 3387, pages 43–54, 2005.

J. Renz and B. Nebel. Efficient methods for qualitative spatial reasoning. *Journal of Artificial Intelligence Research*, 15:289–318, 2001.

A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B. Seo, and Y.-J. Cho. The PEIS-ecology project: vision and results. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008.

F. Tang and L. Parker. ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration. In *Robotics and Automation*, pages 1501–1508. IEEE, 2005.

M. Y. Vardi. Automata-theoretic model checking revisited. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 137–150. Springer, 2007.

F. Wolter and M. Zakharyaschev. Spatio-temporal representation and reasoning based on RCC-8. In *Proceedings of the Seventh Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 3–14, 2000.

Department of Computer and Information Science
Linköpings universitet

Licentiate Theses

Linköpings Studies in Science and Technology
Faculty of Arts and Sciences

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge- Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.
No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.

No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.
No 598 **Rego Granlund:** C^{3}Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel och Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.

No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.

No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.

No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.

No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.

FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.

No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.

No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.

No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.

No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.

No 800 **Anders Subotic:** Software Quality Inspection, 1999.

No 807 **Svein Bergum**: Managerial communication in telework, 2000.

No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.

FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.

No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.

No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.

No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.

No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.

FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.

No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.

No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.

FiF-a 37 **Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.

FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.

FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.

No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.

No 863 **Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.

No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.

No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.

No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue systems, 2001.

FiF-a 47 **Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.

No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.

No 906 **Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.

No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.

No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.

FiF-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.

FiF-a-51 **Per Oscarsson:** Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.

No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.

No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.

No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.

No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.

No 938 **Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.

No 942 **Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.

No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.

FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.

No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.

No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.

No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.

FiF-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.

No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.

No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.

No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.

No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.

No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.

No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002.
No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62 **Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
No 1024 **Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
No 1034 **Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
FiF-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
FiF-a 71 **Emma Eliason:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055 **Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130 **Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138 **Thomas Gustafsson:** Maintaining Data Consistency in Embedded Databases for Vehicular Systems, 2004.
No 1149 **Vaida Jakoniené:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156 **Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172 **Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183 **Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184 **Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185 **Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190 **David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.

No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
No 1192 **Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194 **Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204 **Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.
No 1206 **Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207 **John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209 **Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225 **He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228 **Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229 **Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.
No 1231 **Kalle Burbeck:** Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.
No 1233 **Daniela Mihailescu:** Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.
No 1244 **Jörgen Skågeby:** Public and Non-public gifting on the Internet, 2006.
No 1248 **Karolina Eliasson:** The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.
No 1263 **Misook Park-Westman:** Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.
FiF-a 90 **Amra Halilovic:** Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.
No 1272 **Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.
No 1277 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.
No 1283 **Håkan Hasewinkel:** A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.
FiF-a 91 **Hanna Broberg:** Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.
No 1286 **Robert Kaminski:** Towards an XML Document Restructuring Framework, 2006.
No 1293 **Jiri Trnka:** Prerequisites for data sharing in emergency management, 2007.
No 1302 **Björn Hägglund:** A Framework for Designing Constraint Stores, 2007.
No 1303 **Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.
No 1305 **Magnus Ingmarsson:** Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.
No 1306 **Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.
No 1307 **Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.
No 1309 **Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.
No 1312 **Henrik Svensson**: Embodied simulation as off-line representation, 2007.
No 1313 **Zhiyuan He:** System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.
No 1317 **Jonas Elmqvist:** Components, Safety Interfaces and Compositional Analysis, 2007.
No 1320 **Håkan Sundblad:** Question Classification in Question Answering Systems, 2007.
No 1323 **Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.
No 1329 **Martin Magnusson:** Deductive Planning and Composite Actions in Temporal Action Logic, 2007.
No 1331 **Mikael Asplund:** Restoring Consistency after Network Partitions, 2007.
No 1332 **Martin Fransson:** Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.
No 1333 **Karin Camara:** A Visual Query Language Served by a Multi-sensor Environment, 2007.
No 1337 **David Broman:** Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.
No 1339 **Mikhail Chalabine:** Invasive Interactive Parallelization, 2007.
No 1351 **Susanna Nilsson:** A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.
No 1353 **Shanai Ardi:** A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.
No 1356 **Erik Kuiper:** Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.
No 1359 **Jana Rambusch**: Situated Play, 2008.
No 1361 **Martin Karresand:** Completing the Picture - Fragments and Back Again, 2008.
No 1363 **Per Nyblom:** Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.
No 1371 **Fredrik Lantz:** Terrain Object Recognition and Context Fusion for Decision Support, 2008.
No 1373 **Martin Östlund:** Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.
No 1381 **Håkan Lundvall:** Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.
No 1386 **Mirko Thorstensson:** Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.
No 1387 **Bahlol Rahimi:** Implementation of Health Information Systems, 2008.
No 1392 **Maria Holmqvist:** Word Alignment by Re-using Parallel Phrases, 2008.
No 1393 **Mattias Eriksson**: Integrated Software Pipelining, 2009.
No 1401 **Annika Öhgren:** Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.
No 1410 **Rickard Holsmark:** Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.
No 1421 **Sara Stymne:** Compound Processing for Phrase-Based Statistical Machine Translation, 2009.
No 1427 **Tommy Ellqvist**: Supporting Scientific Collaboration through Workflows and Provenance, 2009.
No 1450 **Fabian Segelström:** Visualisations in Service Design, 2010.
No 1459 **Min Bao**: System Level Techniques for Temperature-Aware Energy Optimization, 2010.
No 1466 **Mohammad Saifullah**: Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011

No 1468 **Qiang Liu**: Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.
No 1469 **Ruxandra Pop**: Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011.
No 1476 **Per-Magnus Olsson**: Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011.
No 1481 **Anna Vapen**: Contributions to Web Authentication for Untrusted Computers, 2011.
No 1485 **Loove Broms:** Sustainable Interactions: Studies in the Design of Energy Awareness Artefacts, 2011.
FiF-a 101 **Johan Blomkvist:** Conceptualising Prototypes in Service Design, 2011.
No 1490 **Håkan Warnquist:** Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis, 2011.
No 1503 **Jakob Rosén:** Predictable Real-Time Applications on Multiprocessor Systems-on-Chip, 2011.
No 1504 **Usman Dastgeer:** Skeleton Programming for Heterogeneous GPU-based Systems, 2011.
No 1506 **David Landén:** Complex Task Allocation for Delegation: From Theory to Practice, 2011.
No 1507 **Kristian Stavåker**: Contributions to Parallel Simulation of Equation-Based Models on Graphics Processing Units, 2011.
No 1509 **Mariusz Wzorek:** Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems, 2011.
No 1510 **Piotr Rudol:** Increasing Autonomy of Unmanned Aircraft Systems Through the Use of Imaging Sensors, 2011.
No 1513 **Anders Carstensen:** The Evolution of the Connector View Concept: Enterprise Models for Interoperability Solutions in the Extended Enterprise, 2011.
No 1523 **Jody Foo:** Computational Terminology: Exploring Bilingual and Monolingual Term Extraction, 2012.
No 1550 **Anders Fröberg:** Models and Tools for Distributed User Interface Development, 2012.
No 1558 **Dimitar Nikolov:** Optimizing Fault Tolerance for Real-Time Systems, 2012.
No 1582 **Dennis Andersson:** Mission Experience: How to Model and Capture it to Enable Vicarious Learning, 2013.
No 1586 **Massimiliano Raciti:** Anomaly Detection and its Adaptation: Studies on Cyber-physical Systems, 2013.
No 1588 **Banafsheh Khademhosseinieh:** Towards an Approach for Efficiency Evaluation of Enterprise Modeling Methods, 2013.
No 1589 **Amy Rankin:** Resilience in High Risk Work: Analysing Adaptive Performance, 2013.
No 1592 **Martin Sjölund:** Tools for Understanding, Debugging, and Simulation Performance Improvement of Equation-Based Models, 2013.
No 1606 **Karl Hammar:** Towards an Ontology Design Pattern Quality Model, 2013.
No 1624 **Maria Vasilevskaya:** Designing Security-enhanced Embedded Systems: Bridging Two Islands of Expertise, 2013.
No 1627 **Ekhiotz Vergara:** Exploiting Energy Awareness in Mobile Communication, 2013.
No 1644 **Valentina Ivanova:** Integration of Ontology Alignment and Ontology Debugging for Taxonomy Networks, 2014.
No 1647 **Dag Sonntag:** A Study of Chain Graph Interpretations, 2014.
No 1657 **Kiril Kiryazov:** Grounding Emotion Appraisal in Autonomous Humanoids, 2014.
No 1683 **Zlatan Dragisic:** Completing the Is-a Structure in Description Logics Ontologies, 2014.
No 1688 **Erik Hansson:** Code Generation and Global Optimization Techniques for a Reconfigurable PRAM-NUMA Multicore Architecture, 2014.
No 1715 **Nicolas Melot:** Energy-Efficient Computing over Streams with Massively Parallel Architectures, 2015.
No 1716 **Mahder Gebremedhin:** Automatic and Explicit Parallelization Approaches for Mathematical Simulation Models, 2015.
No 1722 **Mikael Nilsson:** Efficient Temporal Reasoning with Uncertainty, 2015.
No 1732 **Vladislavs Jahundovics:** Automatic Verification of Parameterized Sytems by Over-Approximation, 2015.
FiF 118 **Camilla Kirkegaard:** Adding Challenge to a Teachable Agent in a Virtual Learning Environment, 2016.
No 1758 **Vengatanathan Krishnamoorthi:** Efficient and Scalable Content Delivery of Linear and Interactive Branched Videos, 2016.
No 1771 **Andreas Löfwenmark:** Timing Predictability in Future Multi-Core Avionics Systems, 2017.
No 1777 **Anders Andersson:** Extensions for Distributed Moving Base Driving Simulators, 2017.
No 1780 **Olov Andersson:** Methods for Scalable and Safe Robot Learning, 2017.
No 1783 **Daniel de Leng**: Spatio-Temporal Stream Reasoning with Adaptive State Stream Generation, 2017.