

Institutionen för Datavetenskap
Department of Computer and Information Science

Master's thesis

**Extending the Stream Reasoning in
DyKnow with Spatial Reasoning in
RCC-8**

by

Daniel Lazarovski

LIU-IDA/LITH-EX-A-12/008-SE

2012-02-10



Linköpings universitet

Institutionen för Datavetenskap
Department of Computer and Information Science

Master's thesis

**Extending the Stream Reasoning in
DyKnow with Spatial Reasoning in
RCC-8**

by

Daniel Lazarovski

LIU-IDA/LITH-EX-A-12/008-SE

2012-02-10

Supervisor: Fredrik Heintz
Department of Computer and Information Science
KPLAB - Knowledge Processing Lab

Examiner: Fredrik Heintz
Department of Computer and Information Science
KPLAB - Knowledge Processing Lab

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ick-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

Abstract

Autonomous systems require a lot of information about the environment in which they operate in order to perform different high-level tasks. The information is made available through various sources, such as remote and on-board sensors, databases, GIS, the Internet, etc. The sensory input especially is incrementally available to the systems and can be represented as streams. High-level tasks often require some sort of reasoning over the input data, however raw streaming input is often not suitable for the higher level representations needed for reasoning. DyKnow is a stream processing framework that provides functionalities to represent knowledge needed for reasoning from streaming inputs. DyKnow has been used within a platform for task planning and execution monitoring for UAVs. The execution monitoring is performed using formula progression with monitor rules specified as temporal logic formulas. In this thesis we present an analysis for providing spatio-temporal functionalities to the formula progressor and we extend the formula progression with spatial reasoning in RCC-8. The result implementation is capable of evaluating spatio-temporal logic formulas using progression over streaming data. In addition, a ROS implementation of the formula progressor is presented as a part of a spatio-temporal stream reasoning architecture in ROS.

Contents

1	Introduction	1
1.1	Background	1
1.2	Goal	3
1.3	Thesis outline	3
2	DyKnow and ROS	4
2.1	DyKnow	4
2.1.1	Basic Concepts	5
2.1.2	Formula Progression	7
2.2	ROS	9
3	Qualitative Spatio-Temporal Reasoning	11
3.1	Constraint Satisfaction	12
3.2	Qualitative Spatial Reasoning	14
3.2.1	Aspects of space	14
3.2.1.1	Orientation	14
3.2.1.2	Distance	17
3.2.1.3	Topology	17
3.2.1.4	Combining multiple aspects of space	19
3.2.2	RCC-8	19
3.3	Qualitative Spatio-Temporal Reasoning	23
3.3.1	Approaches to Spatio-Temporal Reasoning	23
3.3.2	PSTL	25
3.3.2.1	ST_0	26
3.3.2.2	ST_1	26
3.3.2.3	ST_2	27
3.3.2.4	ST_i^+	27
3.4	Reasoning engines	28
3.4.1	GQR	28
4	Analysis	31
4.1	Spatio-temporal stream reasoning in ROS	31
4.2	Stream Reasoner	33
4.3	Spatial Reasoning	34

4.4	Extending the MTL progressor	36
5	Implementation	39
5.1	ROS implementation	39
5.2	Scenario	42
6	Performance Evaluation	54
6.1	Test cases	54
6.2	Test results	55
6.3	Discussion	62
7	Conclusion	64
7.1	Summary	64
7.2	Discussion	65
7.2.1	Stream reasoning perspective	66
7.2.2	Spatio-temporal reasoning perspective	66
7.2.3	UAV domain perspective	67
7.3	Future work	68
	Bibliography	69

List of Figures

2.1	Knowledge Process (from Heintz [2009])	5
2.2	Task planning and execution monitoring overview (from Heintz [2009])	7
2.3	Synchronizing fluent streams in DyKnow (from Heintz [2009])	8
2.4	ROS network	10
3.1	Freksa’s double cross [Freksa, 1992]	15
3.2	Cone-based and projection-based cardinal relations [Ligozat, 1998]	16
3.3	Star calculi (from [Renz and Mitra, 2004])	16
3.4	Rectangle algebra [Balbiani et al., 1998] and direction-relation matrix Goyal and Egenhofer [2000] (from [Renz and Nebel, 2007])	17
3.5	The eight JEPD relations of RCC-8 [Randell et al., 1992] . .	18
3.6	The five JEPD relations of RCC-5	18
3.7	Relational lattice of relations based on $C(x, y)$ (from [Cohn et al., 1997a])	21
3.8	The eight JEPD relations of RCC-8 [Randell et al., 1992] . .	21
3.9	The composition table of RCC-8	22
3.10	The continuity network of RCC-8 [Cohn et al., 1997b]	23
3.11	The six motion classes (from [Muller, 1998])	24
4.1	Overview of the DyKnow architecture for ROS	32
5.1	Scenario environment	43
5.2	Sequence diagram for <code>evaluate_formulas</code> service request . .	48
5.3	Sequence diagram for <code>load_spatial_kb</code> service request	48
5.4	Sequence diagram for handling updates	50
6.1	Average time to progress a state over different numbers of temporal logic formulas	57
6.2	Average time to progress a state over a single formula with varying number of spatial and non-spatial terms	59

6.3	Average time to progress a state with different numbers of regions in the KB	60
6.4	Average time to progress a state when algebraic closure needs to be enforced	61

List of Tables

- 6.1 Average time to progress a state over different numbers of temporal logic formulas 56
- 6.2 Average time to progress a state over a single formula with varying number of spatial and non-spatial terms 58
- 6.3 Average time to progress a state with different numbers of regions in the KB 60
- 6.4 Average time to progress a state when algebraic closure needs to be enforced 61

Listings

3.1	Path-consistency algorithm	29
3.2	Scenario-consistency algorithm	30
5.1	EvaluateFormulas.srv	40
5.2	Domain.msg	40
5.3	Region.msg	41
5.4	RCC8Relation.msg	41
5.5	FieldFeatureType.msg	41
5.6	Sample.msg	42
5.7	Field.msg	42
5.8	FormulaResult.msg	42
5.9	evaluate_formulas service request	44
5.10	Example state sample message	49

Chapter 1

Introduction

The introduction chapter provides overview of the background of the work presented in the thesis report and presents the goals of this thesis project.

1.1 Background

Autonomous systems perform different kinds of high-level functions during run-time, like motion and task planning, event recognition, execution monitoring, etc. Most of these functionalities require some sort of reasoning to be performed. In order to perform the functions, the systems rely on information about the environment in which they operate. The information needed to the autonomous systems is provided as different kinds of input and is usually made available through various on-board or remote sensors, databases, GIS, the Internet etc. Most of the data, especially the sensory input become incrementally available during execution. These data are susceptible to different kinds of noise and usually in the raw numeric form is not appropriate for higher level reasoning.

DyKnow [Heintz, 2009] (section 2.1) is a stream based knowledge processing middleware framework that bridges the gap between raw numerical data available from sensors and higher level knowledge needed for reasoning. It provides support for processing incrementally incoming data, defined as streams and supports varying levels of abstraction.

DyKnow has been used for execution monitoring within a UAV platform [Doherty, Kvarnström, and Heintz, 2009]. The UAV platform uses automated task planning techniques and the execution monitoring system is responsible to check if the execution is going according to the generated plan. The execution monitoring is performed according to specific monitor rules, specified as temporal logic formulas. The temporal logic used, metric temporal logic (MTL) [Koymans, 1990], supports time intervals, meaning that formulas can be defined over specific or infinite time intervals. For example, a monitor rule stating that a UAV must always within the next 30

seconds fly at altitude higher than 20 meters can be specified as a temporal logic formula. As input data becomes incrementally available from the sensors, DyKnow processes the data and evaluates the monitor formulas using formula progression.

DyKnow provides processing of data on different levels of abstraction that can be transformed to higher level knowledge needed for reasoning. Therefore it is possible to support different kinds of reasoning in DyKnow. Two such symbolic reasoning engines are already implemented in DyKnow, a temporal logical progression engine and a chronicle recognition engine for recognizing complex events [Heintz, 2009]. A further reasoning approach applicable in the UAV domain is qualitative spatial and spatio-temporal reasoning. With the support of qualitative spatio-temporal reasoning and representation techniques it would be possible to express and monitor spatial relations between objects. Therefore, it would be beneficial to provide spatial reasoning capabilities to the execution monitoring system, since it would extend the possible scenarios that can be monitored during execution.

There are many different possible spatial and spatio-temporal approaches to choose from, but this thesis project is focused on integrating qualitative spatial reasoning with the Region Connection Calculus (RCC-8) [Randell et al., 1992] (section 3.2.2) with the existing temporal logic formula evaluation, in order to provide qualitative spatio-temporal reasoning based on the semantics of the spatio-temporal modal logic proposed by Bennett, Cohn, Wolter, and Zakharyashev [2002] (section 3.3.2). This would make it possible to express monitor rules that also include spatial relations. For example, there can be monitor rules stating that the UAV must not fly over a restricted region within the next 10 minutes, or that all UAVs must stay over the road within the next 30 seconds. These rules can be then specified as a spatio-temporal monitor formulas.

Although spatial relations can be determined directly from observations of the environment or some form of qualitative data, this might not always be sufficient. In many situations it is not possible to have the full set of observations of the complete spatial environment, since some changes in relations between objects may not be detected. Such cases include for example incomplete or imprecise GIS data, imprecise sensors, incomplete knowledge base of spatial relations. For example if the UAV changes its relation to a car, and to a road which is not detected, then depending on the relation between the car and the road it might be possible from the change of relations between the UAV and the car to determine the new relation between the UAV and the road. This makes it possible to reason about the spatial configurations of the UAVs and the spatial environment when dealing with incomplete and imprecise input from various sources.

This thesis is a part of a research effort to provide an implementation of DyKnow in ROS (Robot Operating System) [Quigley et al., 2009] (section 2.2). ROS is modular, platform and language independent, suitable for large systems. Because of the modularity of ROS it is possible to provide

the different parts of DyKnow as separate modules and to provide further extensions.

1.2 Goal

The goal of the Master's thesis project is to provide spatio-temporal reasoning capabilities to the existing formula progression implementation. Since the original solution is based on temporal logic it is needed to analyze different spatial and spatio-temporal reasoning approaches and select a suitable approach that is compatible with the existing temporal logic and to find a way to integrate it together with the temporal logic formula progressor.

The idea is to integrate the progressor within a spatio-temporal stream reasoning architecture for ROS. Therefore, the additional goal of the thesis project is to provide a ROS implementation of the progressor that can be used as a part of the architecture or as an independent ROS module.

1.3 Thesis outline

The outline of the remainder of the thesis report is as follows:

Chapter 2 presents an overview of the DyKnow framework and the application in automated planning and execution monitoring using formula progression. In addition, an introduction and overview of the basic concepts of ROS is given.

Chapter 3 presents an overview of spatial and spatio-temporal reasoning. Different approaches and calculi found in the literature are presented, with the main focus on RCC-8 and PSTL.

Chapter 4 contains an analysis of the system for spatio-temporal reasoning in ROS and describes our proposed solution for incorporating spatial reasoning into the existing formula progressor.

Chapter 5 describes in detail the ROS implementation of the formula progressor and provides an example scenario.

Chapter 6 presents and discusses the results of the performance evaluation of the formula progressor.

Chapter 7 gives a summary of the thesis project and discusses possible future work directions.

Chapter 2

DyKnow and ROS

This chapter provides overview of the stream based knowledge processing middleware framework DyKnow. The basic concepts and design principles are described and the focus is on the application in execution monitoring of automated plans in the UAV domain, mainly the formula progression.

In addition, an introduction and overview of the basic concepts of Robot Operating System (ROS) is provided in this chapter.

2.1 DyKnow

Autonomous systems that perform various higher level functions rely on information about the environment obtained through various distributed and heterogeneous sources such as on-board or remote sensors, databases, GIS, the Internet etc. Most of these data, mainly those available through sensors often become increasingly available during run-time, so mechanisms capable of handling streaming data are needed. In addition, the data are often noisy and incomplete. Furthermore, these are low-level quantitative data that are not well-fitted for higher level reasoning functionalities. Therefore there is a need to process the data to overcome noise, incompleteness and provide higher level knowledge representations.

Dyknow [Heintz, 2009] is a stream based knowledge processing middleware framework created to integrate components dealing with raw data from sensors, databases or similar, with components that perform higher level reasoning about the environment. Heintz [2009] defines a knowledge processing middleware as a “systematic and principled software framework for bridging the gap between the information about the world through sensing and the knowledge needed when reasoning about the world”.

Dyknow is built on six main design requirements:

- Allow integration of sensory data from heterogeneous and distributed sources that can be processed with various levels of abstraction and can be finally transformed into knowledge that can be used for reasoning.

- Support qualitative and quantitative processing, to deal with quantitative sensory input and qualitative representations needed for higher level reasoning.
- Support both top-down and bottom-up data flow model, to allow interaction in both directions between different dependent abstraction levels.
- Allow uncertainty on different levels of abstraction, to handle different types of uncertainty in raw sensor data, in the symbolic representations of objects, in events and situations, etc.
- Support flexible configurations of the processing that can be dynamically changed or reconfigured to handle changes in the environment
- Support declarative specification of information to specify the desired properties of the processing.

2.1.1 Basic Concepts

The stream-based knowledge processing middleware framework presented by Heintz [2009] is based on *knowledge processes* and *streams*. Knowledge processes (figure 2.1) are models of the distinct functionalities defined within the system. A knowledge process performs processing on information generated by other processes or obtained through other sources and also generates information itself as a result of the processing. Information exchanged between processes becomes incrementally available. To take this into account, the information flow between the processes is modeled as streams. Hence, the inputs and outputs of knowledge processes are in the form of streams.

The stream-based approach presents a suitable mechanism for supporting publisher/subscriber communication model between knowledge processes. A knowledge process can subscribe on a stream to receive the information generated and published by another knowledge process. The knowledge processes publish their results through a *stream generator* that accepts subscriptions and creates streams based on a specific *policy*. The policy specifies the desired properties of the stream.

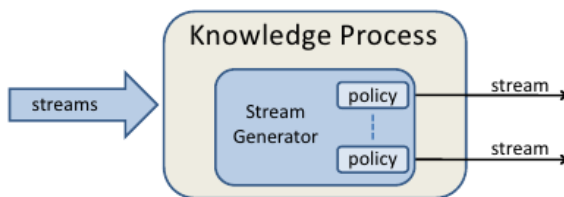


Figure 2.1: Knowledge Process (from Heintz [2009])

Furthermore there are four different types of knowledge processes defined. *Primitive processes* take no streams as input but provide one or more output streams. These processes provide the information from outside world information sources, such as sensors or databases, in the form of streams. *Refinement processes* take one or more streams as input and provide one or more output streams. These processes refine the data from the input stream. *Configuration processes* take one or more streams as input and create and remove knowledge processes and streams. *Mediation processes* have varying number of input streams and a fixed number of output streams. These processes allow dynamic reconfiguration of input streams.

The aforementioned concepts are defined as general requirements for a stream-based knowledge processing middleware, and DyKnow is one such concrete framework. In DyKnow *objects* and *features* are used to model the environment. The world consists of distinct objects that can be concrete or abstract. The features represent the properties of the world and can be attributes of objects or relations between objects. A feature has a value at every time-point and values can change over time. The objects and the values are defined by a *knowledge processing domain*. A knowledge processing domain is defined as a triple $\langle O, T, P \rangle$ where O is a set of objects, T a set of time-points and P a set of primitive values. For a knowledge processing domain $D = \langle O, T, P \rangle$ the set of all possible values is denoted V_D .

In DyKnow streams are conceptualized as *fluent streams* which are defined as streams of *samples*. A sample is a triplet $\langle t_a, t_v, v \rangle$ where $t_a \in T$ is the available time of the sample, $t_v \in T$ is the valid time of the sample and $v \in V_D$ is the value of the sample. The interpretation of a sample is that the value v is an actual or estimated value of a feature, at a time-point t_v and this value is available to the knowledge process at time-point t_a . Therefore, a fluent stream represents the approximation of the value of a feature over time. The set of all possible samples in a knowledge domain D is denoted by S_D and all possible fluent streams, F_D .

The knowledge processes defined for a stream-based knowledge processing middleware, in DyKnow are conceptualized as *sources*, corresponding to primitive processes, and *computational units*, that correspond to refinement processes. A source produces output in form of streams from external data sources, such as sensors or databases. For a knowledge processing domain $D = \langle O, T, P \rangle$, a source is formally defined as a function $T \rightarrow S_D$, mapping time-points to samples. A computational unit has an internal state and has one or more input fluent streams and one output fluent stream. To produce the output only the most recent sample of each input stream is considered and processed. A computational unit with $n > 0$ input streams is formally defined as a mapping from a time-point, n samples and the previous internal state to a sample and a new internal state, in particular as a partial function $T \times S_D^n \times V_D \rightarrow S_D \times V_D$.

2.1.2 Formula Progression

DyKnow has been used as a part of a planing and execution monitoring framework for unmanned aircraft systems [Doherty, Kvarnström, and Heintz, 2009, Heintz, 2009] to provide knowledge processing of information from heterogeneous sources on different levels of abstractions.

The UAS software platform presented by Doherty, Kvarnström, and Heintz [2009] uses a logic-based automated planner, TALPlanner [Kvarnström and Doherty, 2000] to generate mission plans for a given goal specification. In order to deduce a plan that achieves the required goals the planner, like most classical automated planners, works under the assumption that all actions are performed successfully. Therefore, *execution monitoring* is performed to handle the failures in the plan during execution. The conditions to be monitored are specified by temporal logic formulas and are evaluated using *formula progression*. The execution monitor needs to have representations of the state of the system and environment to accurately evaluate monitor formulas. DyKnow is used to continuously send state representations to the execution monitor. The environment is sampled at a certain frequency and a state is generated containing all the features needed for the monitor formulas. The execution monitor can then detect failures as violations of the monitor conditions specified.

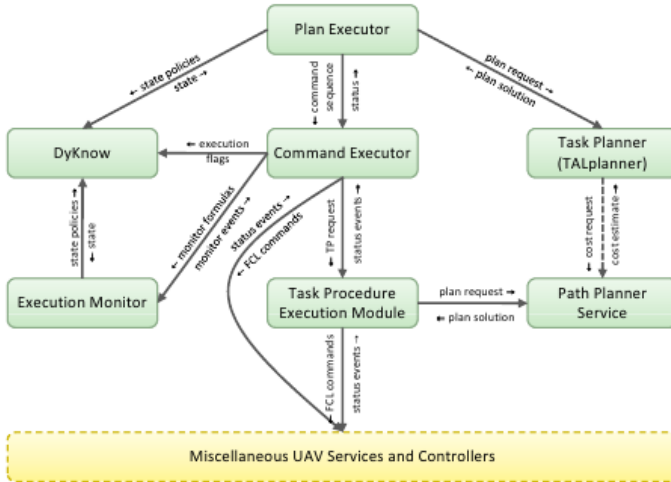


Figure 2.2: Task planning and execution monitoring overview (from Heintz [2009])

The execution monitoring formulas are specified in metric temporal logic (MTL) [Koymans, 1990]. Three tense operators are used for formula progression, U (until), \diamond (eventually) and \square (always). An example monitor formula that specifies that when a UAV is moving faster than a certain min-

imum speed s_{min} , the winch must not be extended above a certain limit w_{min} .

$$\Box \forall uav. speed(uav) > s_{min} \rightarrow winch(uav) \leq w_{min}$$

Such monitor formulas can be specified in the execution monitor and they need to be evaluated to detect failures during execution. This can be achieved by *formula progression*, as new states about the environment are generated by DyKnow, the execution monitor can continuously evaluate the monitor formulas. A formula ϕ that is being progressed is defined to hold for a state sequence $[s_0, s_1, \dots, s_n]$ iff $Progress[\phi, s_0]$ holds for the state sequence $[s_1, \dots, s_n]$. Therefore a formula can be evaluated to true or false as it is being progressed through the states. The progression algorithm used, that progresses formulas over states is based on the one introduced by Bacchus and Kabanza [1996] for metric interval temporal logic (MITL). However, the MITL logic defined and used by Bacchus and Kabanza [1996] is the same as MTL defined by Koymans [1990]. In the remainder of this report, we will use the name MTL.

In order to progress the formulas through states, the states provide values for all the features included in the monitor formulas, for every time-point at which the values are considered to hold. Therefore the monitor formulas are defined on sequences of states [Heintz, 2009]. Heintz [2009] defines a *state* in a knowledge processing domain D as a tuple of values in V_D . Furthermore, a *state sample* is defined as a sample where the value is a state. Finally, a *state stream* is a stream where each stream element is a state sample. The values might come from different and distributed sources as separate fluent streams, however, all values in a state should have the same valid time-point. To achieve this, the incoming fluent streams should be synchronized to generate a state stream where each value in each state has the same valid time (figure 2.3). In DyKnow this is done with a *state synchronization policy* which specifies how the streams are synchronized and defines how state streams are generated.

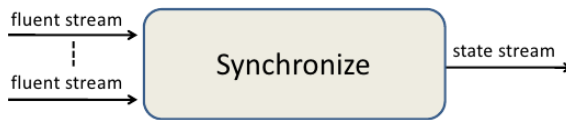


Figure 2.3: Synchronizing fluent streams in DyKnow (from Heintz [2009])

There can be different policies defined depending on the desired properties of the application. One example is to generate states when values have become available in all input streams. Another example is to generate a new state whenever a value becomes available in any input stream, and assume the most recent values for the other fluent streams. Let us assume that a monitor rule related to the speed and altitude of the UAV is specified. Then, in order to evaluate the monitor formula there is a state stream

containing the features $speed(uav1)$ and $altitude(uav1)$. An example state is $\langle 30.2, 27.4 \rangle$ representing the values of $speed(uav1)$ and $altitude(uav1)$ respectively. The data for the speed and altitude of the UAV arrive from two different sensors that are not synchronized. Let us further assume that both fluent streams have sampling periods of 100 time units, but are sampled at different time points, therefore are not synchronized. If we assume that first the information about the speed arrives and next the information about the altitude, there are different state streams that can be generated depending on the selected policy. The first state $\langle s_1, a_1 \rangle$ will be generated after both values become available, and after that states are generated according to the synchronization policy. For example, if a state is generated when a new value becomes available in any fluent stream, the next state is generated when the value for the speed is available, $\langle s_2, a_1 \rangle$, then when the altitude value arrives, $\langle s_2, a_2 \rangle$ and then, $\langle s_3, a_2 \rangle$, $\langle s_3, a_3 \rangle$ and so on. If, for example, a state is generated when values in all state streams become available, after the first state, the state stream will contain the states $\langle s_2, a_2 \rangle$, $\langle s_3, a_3 \rangle$ and so on.

2.2 ROS

The *Robot Operating System*¹[Quigley et al., 2009] (ROS) is a software framework for robotic systems, that “provides a structured communication layer above the host operating systems of a heterogeneous computer cluster”[Quigley et al., 2009]. ROS provides operating system services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, as well as tools and libraries for development and deployment [Conley, 2011]. ROS is free and open-source, and is implemented in different programming languages, such as C++, Python, Lisp with additional experimental libraries implemented in Java and Lua.

The main concepts of ROS are nodes, messages, topics and services.

Nodes are the computation processes used in ROS. As the system is designed to be modular, there can be many nodes. Furthermore, the nodes can reside on different hosts. To utilize this possibility, ROS uses a peer-to-peer topology and allows different ways of communication between nodes via XML-RPC, namely asynchronous publisher/subscriber communication, synchronous request/reply communication and storage on a central location. The *ROS Master* provides node registration and lookup mechanisms so that nodes can find other nodes at runtime. The *Parameter Server* which is used for data storage is part of the ROS Master. The communication between nodes is done with *messages*. A message is a strictly typed data structure which supports primitive types, arrays and nesting. The publisher/subscriber communication architecture is introduced through *topics*.

¹<http://www.ros.org>

Nodes send messages by publishing to a topic, and receive messages by subscribing to a topic. Topics are defined by their names. A node can publish and subscribe to multiple topics, and there can be multiple publishers and subscribers to a topic. For synchronous communication, the request/reply model is provided through *services*. The nodes provide services, and a service is defined by a name, the request message and the response message. As opposed to topics, services use unique names.

The peer-to-peer network is commonly visualized as a graph (figure 2.4) where the ROS nodes are naturally represented as nodes in the graph, and the peer-to-peer links are represented as arcs.

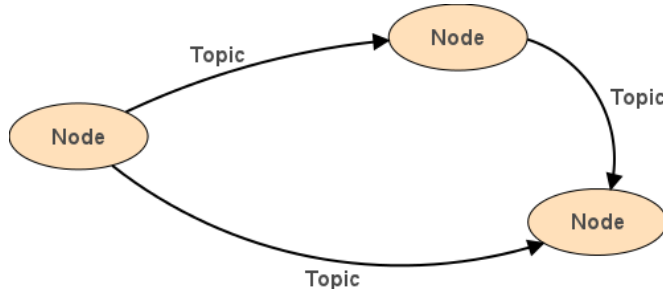


Figure 2.4: ROS network

Chapter 3

Qualitative Spatio-Temporal Reasoning

In this chapter we present the theoretical background of qualitative spatio-temporal reasoning and representation. We present an overview of existing approaches and calculi found in the literature.

Qualitative knowledge represents the features and characteristics that are unique or specific for the aspect of the world under consideration, whereas *quantitative* knowledge represents numerical values according to a specified measurement unit [Hernández, 1994].

Qualitative knowledge is often used in AI for representing different aspects of the world. As the aspects of the world, such as time, space, size, shape, quantity are continuous, different qualitative representations can be specified with different levels of granularity and expressiveness. Hence, qualitative reasoning can be performed with less information than when dealing with purely quantitative knowledge. When performing reasoning about the environment, in robotic systems for example, sometimes less precise data is available from sensors, or less precise results are sufficient, therefore qualitative representation and reasoning is applicable to such scenarios.

We start this chapter with an introduction of constraint satisfaction methods as these are commonly used for qualitative reasoning. Next, we provide an overview on qualitative spatial reasoning (QSR) and the different aspects of space which QSR approaches are based on, focusing on the calculus RCC-8. The next chapter presents existing approaches for qualitative spatio-temporal stream reasoning (QSTR) with focus on PSTL. Finally, an overview of existing spatial reasoning engines is presented.

3.1 Constraint Satisfaction

Constraint-based (constraint satisfaction) methods are widely used for qualitative reasoning. This section gives an introduction to constraint satisfaction methods and techniques relevant for the remainder of this work. More detail on constraint satisfaction can be found in Dechter [2003], Russell and Norvig [2003] and particularly for the case of qualitative spatial and temporal reasoning in [Renz, 2002, Renz and Nebel, 2007].

A *constraint satisfaction problem* (CSP) is defined as a set of variables \mathcal{X} over a domain \mathcal{D} and a set of constraints \mathcal{C} . An *n*-ary *constraint* is represented by a *n*-tuple of variables $\langle x_1, \dots, x_n \rangle$ and a *n*-ary relation $R \subseteq \mathcal{D}^n$. An *instantiation* (evaluation, assignment) of the variables is defined as a function from the set of variables to the domain of values $f : \mathcal{X} \rightarrow \mathcal{D}$. The instantiation f *satisfies the constraint* $R(x_1, \dots, x_i)$ iff $\langle f(x_1), \dots, f(x_n) \rangle \in R$. A *solution* is an instantiation that satisfies all the constraints from \mathcal{C} . A CSP is *consistent* if it has a solution.

In our work we focus on CSPs with binary constraints. Binary CSPs can be represented by a directed graph where each node is a variable and each edge is a relation (binary constraint). This graph is also known as *constraint network*. When expressing binary constraints and relations we will use prefix notation ($R(x_i, x_j)$) or in some cases infix notation ($x_i R x_j$).

A solution to a CSP can be found by backtracking over the domain of values that can be assigned to variables. Since backtracking is exponential in the number of variables, in order to improve it it is possible to apply methods such as ordering the domain values and constraint propagation. The main idea of these methods is to reduce the CSP to an equivalent one that is easier to solve. Ordering the values dictates the order in which they are assigned which influences the branching and pruning of the search tree. Constraint propagation methods enforce various levels of local consistency, thus reducing the domain of the variables and the size of the search space. However, domains can be infinite as it is usually the case with spatial and temporal variables, and then backtracking over the domain is not possible. In the case of infinite domain, a possible approach is to formulate the binary CSP as a *relation algebra*, meaning to use constraints over a finite set of binary relations [Renz and Nebel, 2007].

A relation algebra is an algebraic structure that consists of the following:

- set of binary relations \mathcal{R}
- operations on relations: *union, intersection, complement, composition and conversion*
- relational constants: *empty relation, universal relation and identity relation*

The set of relations \mathcal{R} is closed under the defined operations on relations. The composition relation (\circ) is defined for relations $R, S \in \mathcal{R}$ as follows:

$$R \circ S = \{\langle x, y \rangle \mid \exists z : \langle x, z \rangle \in R \wedge \langle z, y \rangle \in S\}$$

The composition relation is of particular interest as it is used in the *path consistency algorithm* for achieving local consistency. In particular, the path consistency algorithm enforces *path consistency* of a CSP. A CSP is path consistent if for every instantiation of two variables that satisfies the constraints, there exists an instantiation of every third variable that satisfies the constraints between the three variables. Path consistency is enforced by successively applying the following operation to all variables $x_i, x_j, x_k \in \mathcal{X}$ until a fixed point is reached:

$$\forall k : R(x_i, x_j) = R(x_i, x_j) \cap (R(x_i, x_k) \circ R(x_k, x_j))$$

If the result is the empty relation, then the CSP is inconsistent. Otherwise, the resulting CSP is path consistent. Furthermore, the resulting CSP has the same set of solutions as the original one.

As mentioned before, relation algebras are based on a finite set of binary relations. In qualitative spatial and temporal representation and reasoning approaches, these relations usually are *jointly exhaustive and pairwise disjoint* (JEPD), meaning that any pair of entities from the domain are related by exactly one of these JEPD relations. The JEPD relations are also referred to as *atomic relations* or *base relations*. We denote the set of base relations as \mathcal{B} , and then the set of available relations \mathcal{R} is the powerset of \mathcal{B} , $\mathcal{R} = 2^{\mathcal{B}}$. The powerset contains all possible disjunctions of relations of \mathcal{B} , therefore it is possible to represent indefinite knowledge by disjunctions (unions) of base relations. For example, $x\{R_i, R_j, R_k\}y$ states that between x and y exactly one of the base relations R_i, R_j, R_k holds.

For qualitative spatial calculi, the domains are infinite and not well structured. Therefore it is not feasible to compute the composition, instead a *weak composition* can be used, which is the strongest relation that contains the real composition [Renz and Ligozat, 2005]. Formally, weak composition is defined as follows:

$$S \diamond T = \{R_i \in \mathcal{B} \mid R_i \cap (S \circ T) \neq \emptyset\}$$

In some cases weak composition is equal to composition, however when it is not the path consistency algorithm cannot be applied anymore. The *algebraic closure algorithm* has the composition operator replaced with the weak composition operator and enforces algebraic closure, or also called a-closure, on a network [Renz and Ligozat, 2005]. In qualitative spatial and temporal calculi, the (weak) composition of base relations is computed using the formal semantics of the relations. The (weak) composition table is usually pre-computed.

A *scenario* is a constraint network where all constraints are base relations. As the constraints can be unions of base relations, in order to find

a scenario, the constraint network should be refined. Relation R is a *refinement* of relation S iff $R \subseteq S$. Refinement also applies for constraints and sets of constraints. A set of constraints C' is a refinement of C iff both constraint networks have the same set of variables and for all relations R' in C' and all relations R in C constraining the same variables as R' , we have $R' \subseteq R$. If both C and C' are consistent then the refinement C' is called *consistent refinement*. Then, a *consistent scenario* is defined as a scenario that is a consistent refinement [Renz and Nebel, 2007]. If a consistent scenario exists, then the original constraint network is consistent. For more detailed discussion see [Renz and Nebel, 2007].

3.2 Qualitative Spatial Reasoning

The qualitative approach is suitable for expressing spatial knowledge, as describing the spatial features of objects in natural language produces qualitative expressions.

There are different approaches regarding the ontology for qualitative spatial representation and reasoning, however most theories are based either on points or spatial regions as basic spatial entities [Vieu, 1997]. In addition, there are multiple aspects of space which can be described in a qualitative manner, such as, distance, orientation, topology, size, shape. However, most of the work in qualitative spatial reasoning and representation is mainly focused on a single aspect of space, most commonly topology, orientation and distance. With respect to the chosen aspect of space it is common to express the relationships between the spatial entities with qualitative spatial relations which are called *base relations*. To be possible to use constraint based reasoning methods, it is necessary the base relations to be jointly exhaustive and pairwise disjoint (JEPD), meaning that exactly one of these relations holds for any pair of spatial entities. Reasoning can then be performed by using composition of the relations. Therefore it is necessary to provide a composition table which is usually pre-computed [Renz and Nebel, 2007].

3.2.1 Aspects of space

As mentioned before, there are multiple aspects of space, but most of the calculi are focused on a single aspect. In this section we present a short overview of common and well-known approaches for different aspects of space.

3.2.1.1 Orientation

Representing qualitative orientation relations and reasoning about orientation has received a lot of attention in the QSR community. Therefore, many different calculi have been developed.

Qualitative approach is appropriate for expressing orientation relations and for reasoning about orientation. In natural language and communication

orientation between spatial entities is expressed in qualitative terms, such as, “south of”, “behind”, “to the right of”, etc.

Most approaches dealing with orientation and direction use points as basic spatial entities in 2D space. [Renz and Nebel, 2007, Cohn and Renz, 2008].

When it comes to expressing the qualitative spatial relations describing the orientation of one object to a reference object, the frame of reference should also be specified. Thus, the qualitative relations specifying orientation properties are ternary, depending on the primary object, the reference object and the frame of reference. One such approach, using ordering information introduced by Schlieder [1993, 1995], uses three qualitative relations +, - and 0 representing anticlockwise, clockwise and collinear relations respectively for the ordering of triples of points. This ordering information approach is further used in [Schlieder, 1995] for qualitative shape representation. Another approach is the double-cross calculus introduced by Freksa [1992] (figure 3.1). It consists of three axes that define 15 ternary base relations.

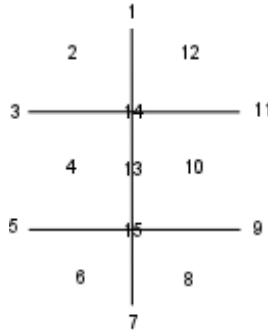


Figure 3.1: Freksa’s double cross [Freksa, 1992]

However, most approaches use a predetermined frame of reference. In this case, it is possible to express the relationship between two objects as a binary relation with respect to the specified frame of reference. Frank [1991] introduced cone-based and projection-based approaches for reasoning about cardinal directions, N , E , S , W or also including NE , SE , SW , NW depending on granularity. The projection-based method is also known as *cardinal relation algebra* [Ligozat, 1998] and uses 9 JEPD relations N , E , S , W , NE , SE , SW , NW , EQ (figure 3.2). A generalisation of this approach, the *Star calculus* was introduced by Renz and Mitra [2004] proposing the use of arbitrary level of granularity (figure 3.3). Another calculus with support for different levels of granularity is the *Orientation Point Relation Algebra* ($OPRA_m$) [Moratz et al., 2005] and it is based on oriented points.

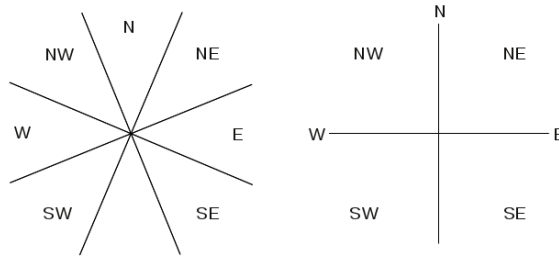


Figure 3.2: Cone-based and projection-based cardinal relations [Ligozat, 1998]

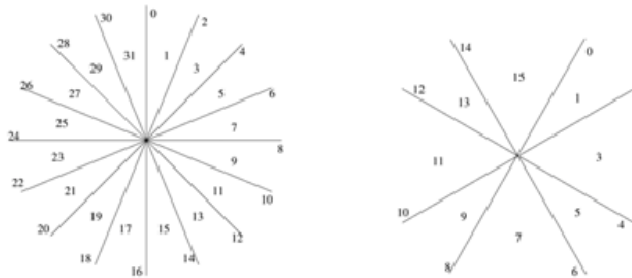


Figure 3.3: Star calculi (from [Renz and Mitra, 2004])

As mentioned earlier, most of the approaches are based on points as basic spatial entities, as it is considered to be difficult to qualitatively express relations between extended regions [Renz and Nebel, 2007, Vieu, 1997]. However, there are some approaches that are based on regions as basic spatial entities. The *rectangle algebra* [Balbiani et al., 1998] (figure 3.4) and the *minimal bounding rectangle* approach [Papadias and Theodoridis, 1997] are based on rectangle regions with sides parallel to the axes. The *direction-relation matrix* approach [Goyal and Egenhofer, 2000] and the *Cardinal Direction Calculus (CDC)* [Goyal and Egenhofer, 2001, Skiadopoulos and Koubarakis, 2005] also use a minimal bounding rectangle, such as the extended lines of the rectangle sides form 9 sectors (figure 3.4). A detailed overview of reasoning about cardinal directions between extended objects is available in [Liu et al., 2010].

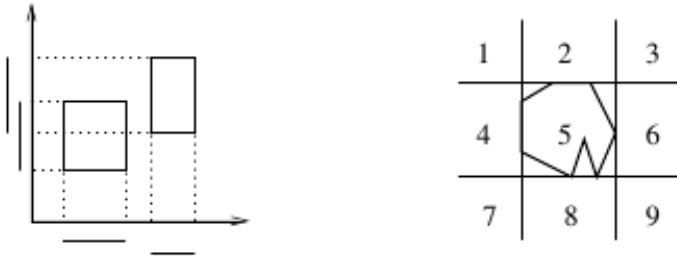


Figure 3.4: Rectangle algebra [Balbiani et al., 1998] and direction-relation matrix Goyal and Egenhofer [2000] (from [Renz and Nebel, 2007])

3.2.1.2 Distance

Qualitative distance has received less attention in the literature and is usually considered in combination with orientation [Vieu, 1997]. Distance is commonly specified as a quantitative value, however qualitative relations about distance are also used, such as absolute binary relations like “far/close to” and relative ternary relations “farther/closer than”. Absolute distance can be expressed with different levels of granularity. When dealing with relative relations the orientation should be taken in consideration. For example, if A is far from B and B is far from C , then the distance between A and C depends on the angle between AB and BC . The combination of distance and orientation is called *positional* information [Renz and Nebel, 2007]. More detailed overviews of approaches regarding qualitative reasoning about distance can be found in [Vieu, 1997, Cohn and Hazarika, 2001, Renz, 2002, Renz and Nebel, 2007, Cohn and Renz, 2008].

3.2.1.3 Topology

Topology is the aspect of space that has received the most attention in the QSR community. Describing topological relationships can only be done in a purely qualitative manner.

Although topology has been studied within mathematics, mathematical topology theories are too abstract and deemed undesirable for qualitative spatial reasoning, therefore when it comes to QSR, the focus is mostly on mereotopology which integrates topology and mereology [Cohn and Renz, 2008].

Most topological approaches use spatial regions as basic spatial entities and are based on Clarke’s *calculus of individuals* [Clarke, 1981] which uses as a basis a single binary relation $C(x, y)$, meaning “ x connects with y ”. Different approaches use different interpretations of the $C(x, y)$ relation based on the views on open, semi-open and closed regions. While Clarke [1981] and Asher and Vieu [1995] differentiate between open and closed regions, Randell, Cui, and Cohn [1992] in their *Region Connection Calculus* (RCC)

reject that distinction. Therefore, in RCC only the topological closure of regions is considered, thus the interpretation of the $C(x, y)$ relation in the RCC theory differs from the one in the calculus of individuals. While in Clarke’s approach “connected” is interpreted that the regions share a point, the interpretation in the RCC theory is that the topological closures of the regions share a point [Cohn et al., 1997a]. Using the $C(x, y)$ relation, many different relations can be defined. Eight of those, *disconnected* (DC), *externally connected* (EC), *partially overlaps* (PO), *equal* (EQ), *tangential proper part* (TPP), *nontangential proper part* ($NTPP$), *tangential proper part inverse* ($TPPi$) and *nontangential proper part inverse* ($NTPPi$) are jointly exhaustive and pairwise disjoint [Randell et al., 1992] and are denoted as RCC-8 (figure 3.5). RCC-5 is another well known approach based on the RCC theory, which uses five base relations, in particular DC and EC are collapsed into *discrete from* (DR), TPP and $NTPP$ are collapsed into *proper part* (PP) and finally, $TPPi$ and $NTPPi$ are collapsed into *proper part inverse* (PPi) (figure 3.6). For more details about RCC-8, see section 3.2.2.

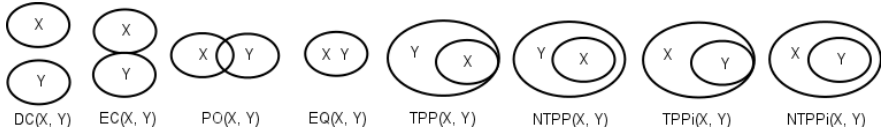


Figure 3.5: The eight JEPD relations of RCC-8 [Randell et al., 1992]

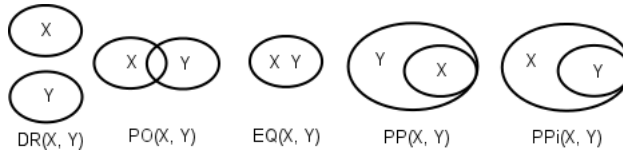


Figure 3.6: The five JEPD relations of RCC-5

Another approach, the *9-intersection* model presented by Egenhofer [1991], considers the interior, complement (exterior) and boundary of a region and describes the topological relations between two regions by the nine possible intersections of the boundaries, interiors and complements of the two regions. This leads to the same set of eight base topological relations as RCC-8.

Vieu [1997] considers the inability to have different kinds of contact in RCC as a drawback introduced by rejecting the distinction of open and closed regions. However, Randell et al. [1992] argue that regions (physical objects) occupying the same locations should not be distinct and that the topological structure containing open, semi-open and closed regions is too rich for their purposes.

3.2.1.4 Combining multiple aspects of space

Although most of the work in QSR has been done with focus on a single aspect of space, in reality different aspects of space cannot always be treated independently. Therefore, there have been efforts for combining more than one aspect of space. As mentioned before, commonly combined aspects are orientation and distance and some of the presented orientation calculi integrate these two aspects.

Gerevini and Renz [2002] introduce an approach that integrates qualitative and quantitative information about size of regions with qualitative topological information based on RCC-8.

Another approach by Liu et al. [2009] combines topology and orientation. It is based on RCC-8 for representing qualitative topological information and the Rectangle Algebra (RA) and Cardinal Direction Calculus (CDC) for representing qualitative orientation information for extended spatial entities.

Renz and Nebel [2007] provide detailed discussion about combining topological and size information, and also topological and directional information for intervals.

3.2.2 RCC-8

RCC-8 is the best known and most widely used and studied qualitative spatial reasoning approach [Renz, 2002]

As mentioned before, RCC [Randell et al., 1992] uses a single binary “connected” relation to define mereotopological relations. The interpretation of the relation $C(x, y)$ is that the topological closures of the regions x and y share a common point. The relation $C(x, y)$ is reflexive and symmetric and therefore two axioms are introduced:

$$\forall x C(x, x) \tag{3.1}$$

$$\forall xy [C(x, y) \Rightarrow C(y, x)] \tag{3.2}$$

Using $C(x, y)$ [Randell et al., 1992] define a basic set of binary relations and provides a formal definition. The relations are the following: $DC(x, y)$ (x is disconnected from y), $P(x, y)$ (x is part of y), $PP(x, y)$ (x is proper part of y), $x = y$ or as it is more commonly used $EQ(x, y)$ (x is identical with y), $O(x, y)$ (x overlaps y), $DR(x, y)$ (x is discreet from y), $PO(x, y)$ (x partially overlaps y), $EC(x, y)$ (x is externally connected to y), $TPP(x, y)$ (x is a tangential proper part of y) and $NTPP(x, y)$ (x is a nontangential proper part of y). The relations P , PP , TPP and $NTPP$ are non-symmetrical and their inverses have been defined as P^{-1} , PP^{-1} , TPP^{-1} , $NTPP^{-1}$. Note that through the text the inverse relations will be denoted as Pi , PPi , $TPPi$ and $NTPPi$. The formal definition of the relations is presented:

$$DC(x, y) \equiv_{def} \neg C(x, y) \quad (3.3)$$

$$P(x, y) \equiv_{def} \forall [C(z, x) \Rightarrow C(z, y)] \quad (3.4)$$

$$PP(x, y) \equiv_{def} P(x, y) \wedge \neg P(y, x) \quad (3.5)$$

$$EQ(x, y) \equiv_{def} P(x, y) \wedge P(y, x) \quad (3.6)$$

$$O(x, y) \equiv_{def} \exists z [P(z, x) \wedge P(z, y)] \quad (3.7)$$

$$PO(x, y) \equiv_{def} O(x, y) \wedge \neg P(x, y) \wedge \neg P(y, x) \quad (3.8)$$

$$DR(x, y) \equiv_{def} \neg O(x, y) \quad (3.9)$$

$$TPP(x, y) \equiv_{def} PP(x, y) \wedge \exists z [EC(z, x) \wedge EC(z, y)] \quad (3.10)$$

$$EC(x, y) \equiv_{def} C(x, y) \wedge \neg O(x, y) \quad (3.11)$$

$$NTPP(x, y) \equiv_{def} PP(x, y) \wedge \neg \exists z [EC(z, x) \wedge EC(z, y)] \quad (3.12)$$

$$Pi(x, y) \equiv_{def} P(y, x) \quad (3.13)$$

$$PPi(x, y) \equiv_{def} PP(y, x) \quad (3.14)$$

$$TPPi(x, y) \equiv_{def} TPP(y, x) \quad (3.15)$$

$$NTPPi(x, y) \equiv_{def} NTPP(y, x) \quad (3.16)$$

In addition, a relational lattice for the set of relations is presented in figure 3.7. The relations in the lattice are ordered such that the most general (weakest) relations are on the top, and the most specific (strongest) relations are at the bottom. For example, TPP implies PP , $NTPP$ also implies PP , and PP implies TPP or $NTPP$. The lattice corresponds to a set of theorems that the authors have verified, such as:

$$\forall xy [PP(x, y) \iff [TPP(x, y) \vee NTPP(x, y)]] \quad (3.17)$$

An additional axiom is also added to the theory, that states that every region has a nontangential proper part:

$$\forall x \exists y [NTPP(y, x)] \quad (3.18)$$

Furthermore, using the relations the authors define the following Boolean functions: $sum(x, y)$, the sum of x and y ; Us , the universal (spatial) region; $compl(x)$, the complement of x ; $prod(x, y)$, the product (intersection) of x and y and $diff(x, y)$, the difference of x and y . From these functions, additional relations can then be defined. Introducing the sum function leads to possibility of having regions that consist of disconnected parts Cohn et al. [1997a] and it can be tested with the predicate:

$$CON(x) \equiv_{def} \forall yz [EQ(sum(y, z), x) \Rightarrow C(y, z)] \quad (3.19)$$

[Cohn et al., 1997a] discuss possibilities to define more relations based on $C(x, y)$ in order to be able to describe shapes of regions (such as doughnut, torus, etc). Furthermore, as the topological representations of the regions

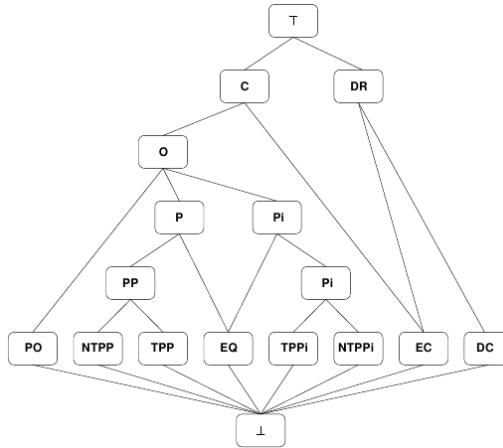


Figure 3.7: Relational lattice of relations based on $C(x, y)$ (from [Cohn et al., 1997a])

are considered to be convex, they introduce primitives in order to be able to create more expressive languages. The *convex hull* primitive $conv(x)$ denotes the convex hull of a region x . A convexity predicate, stating that a region is convex region if it is equal to its convex hull, is then defined:

$$CONV(x) \equiv_{def} EQ(x, conv(x)) \tag{3.20}$$

As mentioned earlier, out of the relations defined by [Randell et al., 1992], there are eight provably JEPD relations that are referred to as RCC-8. These eight relations are DC , EC , PO , EQ , TPP , $NTPP$, $TPPi$ and $NTPPi$ (figure 3.8). In addition to the JEPD relations, in order to be able to perform reasoning using constraint-based methods it is necessary to specify the compositions between the base relations. The compositions are specified as a composition table and are obtained using the formal semantics of the relations. However, Düntsch et al. [2001] show that the composition for RCC-8 is actually weak composition, and the composition table is weak composition table. The composition table of RCC-8 is shown in figure 3.9 (the * symbol denotes the universal relation). From the composition table, given two relations $R_1(x, y)$ and $R_2(y, z)$, the possible relations between x and z can be determined.

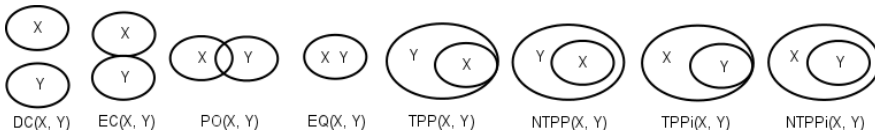


Figure 3.8: The eight JEPD relations of RCC-8 [Randell et al., 1992]

	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ
DC	*	DC,EC,PO, TPP,NTPP	DC,EC,PO, TPP,NTPP	DC,EC,PO, TPP,NTPP	DC,EC,PO, TPP,NTPP	DC	DC	DC
EC	DC,EC,PO, TPPi,NTPPi	DC,EC,PO, TPP,TPPi,EQ	DC,EC,PO, TPP,NTPP	EC,PO,TPP, NTPP	PO,TPP,NTPP	DC,EC	DC	EC
PO	DC,EC,PO, TPPi,NTPPi	DC,EC,PO, TPPi,NTPPi	*	PO,TPP,NTPP	PO,TPP,NTPP	DC,EC,PO, TPPi,NTPPi	DC,EC,PO, TPPi,NTPPi	PO
TPP	DC	DC,EC	DC,EC,PO, TPP,NTPP	TPP,NTPP	NTPP	DC,EC,PO, TPP,TPPi,EQ	DC,EC,PO, TPPi,NTPPi	TPP
NTPP	DC	DC	DC,EC,PO, TPP,NTPP	NTPP	NTPP	DC,EC,PO, TPP,NTPP	*	NTPP
TPPi	DC,EC,PO, TPPi,NTPPi	EC,PO,TPPi, NTPPi	PO,TPPi,NTPPi	PO,TPP,TPPi, EQ	PO,TPP,NTPP	TPPi,NTPPi	NTPPi	TPPi
NTPPi	DC,EC,PO, TPPi,NTPPi	PO,TPPi,NTPPi	PO,TPPi,NTPPi	PO,TPPi,NTPPi	PO,TPP,NTPP, TPPi,NTPPi,EQ	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

Figure 3.9: The composition table of RCC-8

Example RCC-8 Let us assume an environment consisting of a road, a property on the side of the road and a forest on the other side of the road with a restricted region in the forest. Let us then assume that there is a UAV above the property. The spatial configuration of the environment can be represented by the following relations:

$$\begin{aligned}
& EC(\text{road}, \text{property}) \\
& EC(\text{road}, \text{forest}) \\
& DC(\text{property}, \text{forest}) \\
& NTPP(\text{uav}, \text{property}) \vee TPP(\text{uav}, \text{property}) \\
& NTPP(\text{restricted}, \text{forest})
\end{aligned}$$

From the specified relations, using the RCC-8 composition table and the path-consistency algorithm can be deduced that the relation between the UAV and the restricted region is:

$$DC(\text{uav}, \text{restricted})$$

□

These relations are encoded in first-order logic, therefore reasoning about these relations is undecidable. However, [Bennett, 1994, 1996] presents zero-order modal encoding of the RCC-8 relations, thus proves that reasoning over the RCC-8 relations is decidable. In particular, constants represent regions and logical operators represent operations on regions. Furthermore, Renz and Nebel [1999] show that path consistency decides consistency for RCC-8. In addition, as weak composition differs from composition in RCC-8, algebraic closure decides RCC-8 [Renz and Ligozat, 2005].

Cohn et al. [1997b] introduce a set of envisioning axioms for specifying temporal continuity in RCC-8. With this set of axioms it is possible to reason about changes in spatial relations. Namely, the axioms specify which transitions between the spatial relations are possible. The listing of the axioms is given in Cohn et al. [1997b], however these can be represented as a continuity network (conceptual neighborhood graph) as in figure 3.10. For example, if for regions x and y we have $EC(x, y)$, from the continuity network we can observe that through continuous change of the regions, their relation may remain the same or change to $DC(x, y)$ or $PO(x, y)$.

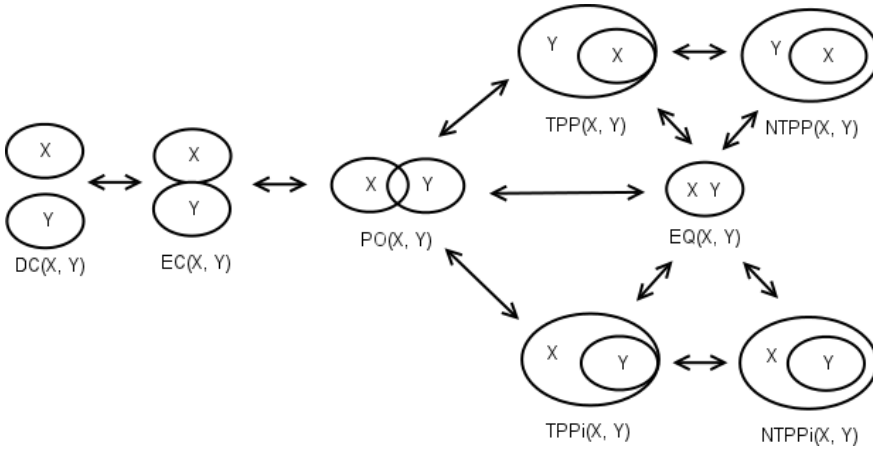


Figure 3.10: The continuity network of RCC-8 [Cohn et al., 1997b]

There are different extensions of the RCC theory, such as RCC-23 [Cohn et al., 1997a], which is concerned with concave regions. Jihong et al. [2007] further refine RCC-23 to RCC-62, a more expressive calculus for representing and reasoning about concave regions.

3.3 Qualitative Spatio-Temporal Reasoning

Qualitative spatio-temporal representation and reasoning (QSTR) is concerned with combining spatial and temporal reasoning. In particular this means reasoning about spatial change. There are different approaches such as reasoning about motion, or approaches concerned with different aspects of space that change over time, such as change in shape, size, position, topology. We are interested in continuous change of topological relations between regions.

3.3.1 Approaches to Spatio-Temporal Reasoning

When it comes to combining these spatial and temporal reasoning, Wolter and Zakharyashev [2003] propose a “naïve” approach which suggests merg-

ing the spatial logic and the temporal logic into a “spatio-temporal hybrid” that provides the desired level of interaction between space and time. Furthermore, they suggest that the construction of the spatio-temporal logic should be based either on the syntactical properties, meaning the axioms of the temporal and the spatial logic are joined together, or based on the semantical properties, meaning the intended models of the temporal and spatial logic are integrated into a multi-dimensional spatio-temporal structure.

In the literature, different spatio-temporal theories can be found. Muller [1998, 2002] presents a qualitative theory of motion of spatial entities. It extends the spatial theory by Asher and Vieu [1995] with temporal order, introduces temporal and spatio-temporal relations and presents six motion classes: *leave*, *hit*, *reach*, *external*, *internal*, *cross* (figure 3.11). Another theory of motion is introduced by Ibrahim and Tawfik [2007] and is based on RCC-8. This approach, uses intervals and determines the motion class based on the RCC-8 relations that hold at the start and at the end of the interval. It presents nine motion classes: *leave*, *reach*, *hit*, *split*, *peripheral*, *expand*, *shrink*, *internal*, *external*. Furthermore they define the conceptual neighbors for the classes and present a composition table for the motion classes.

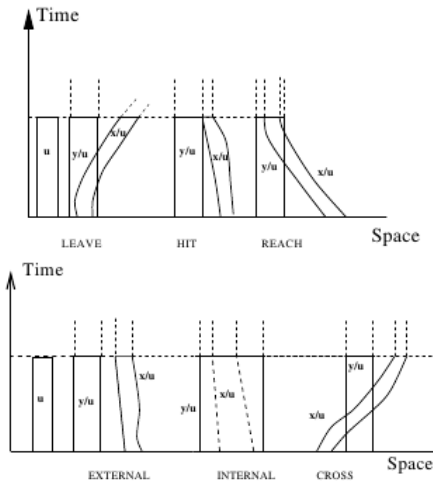


Figure 3.11: The six motion classes (from [Muller, 1998])

Other approaches explore the use of multi-dimensional modal logic for spatio-temporal reasoning [Bennett and Cohn, 1999, Wolter and Zakharyashev, 2000, Bennett et al., 2002, Wolter and Zakharyashev, 2003]. Bennett and Cohn [1999] discuss possible uses of multi-dimensional and in particular 2D modal logic for spatio-temporal reasoning, planar space and continuous motion. In the case of spatio-temporal reasoning, they discuss the idea of using bi-modal logic, where one modal operator is used as a spatial modality and one modal operator as a temporal modality. Wolter and Zakharyashev

[2000] introduce languages for spatio-temporal representation and reasoning about topological relations that change over time, the ST_i languages. In this approach RCC-8 is combined with the point based propositional temporal logic (PTL). For more details, see section 3.3.2. Bennett et al. [2002] and Wolter and Zakharyashev [2003] present further extensions of the approach, in which RCC-8 is combined with branching time and interval time based on Allen's interval algebra. Gerevini and Nebel [2002] present a similar approach for spatio-temporal reasoning that integrates RCC-8 with Allen's interval algebra, with the addition of size persistence constraint and continuity constraint.

3.3.2 PSTL

Propositional spatio-temporal logic (PSTL) is a Cartesian product of spatial and temporal structures [Wolter and Zakharyashev, 2003], namely the temporal logic PTL and the modal logic $S4_u$ [Bennett et al., 2002]. In particular, $S4_u$ is used to encode the topological relations of RCC-8. In this section the focus is on the expressiveness and the semantics of the spatio-temporal logic, for details on the modal encoding and the formal theory refer to Wolter and Zakharyashev [2000], Bennett et al. [2002] and Wolter and Zakharyashev [2003].

The point based propositional temporal logic (PTL) is based on the flow of time $\langle \mathbb{N}, < \rangle$ and uses the binary temporal operators S (since) and U (until), with the intended meaning [Wolter and Zakharyashev, 2003]:

- $vU\varphi$ (v holds until φ holds)
- $vS\varphi$ (v has been true since φ was true)

Based on the S and U operators, other standard operators can be defined [Bennett et al., 2002]:

- $\bigcirc^+\varphi \equiv_{def} \varphi U \varphi$ (at the next moment φ)
- $\diamond^+ \equiv_{def} (p \vee \neg p) U \varphi$ (at some time in the future φ)
- $\square^+ \equiv_{def} \neg \diamond^+ \neg \varphi$ (always in the future φ)

The past counterparts \bigcirc^- , \diamond^- and \square^- can be defined using the S operator. Note that in the remainder of the report we use \bigcirc , \diamond and \square instead of \bigcirc^+ , \diamond^+ and \square^+ respectively.

Wolter and Zakharyashev [2000] present different ways of introducing a temporal dimension into the syntax of RCC-8, namely the spatio-temporal languages ST_i . There are three different languages, based on the different level of integration of the temporal operators into the RCC-8 relations.

3.3.2.1 ST_0

The language ST_0 allows applications of the temporal operators S and U and Booleans to spatial formulas. In this way it is possible to express the changes of the spatial relations between the regions over time. For example:

$$\diamond NTPP(x, y) \quad (3.21)$$

$$\square DC(uav, restricted) \quad (3.22)$$

$$\square PP(tracked_obj, visible_region(uav)) \quad (3.23)$$

The formula 3.21 means that the region x will eventually become non-tangential proper part of the region y . The formula 3.22 means that the UAV will always be disconnected from the restricted region, while the formula 3.23 means that the tracked objects is always within the visible region of the UAV.

Furthermore, it is possible to express the transitions within the continuity network of RCC-8 presented in figure 3.10 [Wolter and Zakharyashev, 2000] for example:

$$\square(EC(x, y) \Rightarrow \bigcirc(DC(x, y) \vee EC(x, y) \vee PO(x, y)))$$

Meaning that always two externally connected (EC) regions in the next time-step remain either in the same relation or become disconnected (DC) or partially overlapping (PO). Such expressions can be formulated for all transitions between the relations.

Since ST_0 allows to apply the temporal operators only to spatial formulas, it is only possible to express the relations of the regions at the same point in time. In reality, however, the regions themselves can change over time.

3.3.2.2 ST_1

The language ST_1 is more expressive, in a way that it extends ST_0 by allowing application of the next-time operator \bigcirc to region variables, therefore in ST_1 it is possible to express spatial relations between the extensions of regions over time. For example, it is possible to express that the region x will always be the same (formula 3.24), or that it will never shrink (formula 3.25) [Wolter and Zakharyashev, 2003]:

$$\square EQ(x, \bigcirc x) \quad (3.24)$$

$$\square P(x, \bigcirc x) \quad (3.25)$$

Another example formula that can be expressed:

$$\square(EQ(\bigcirc uav_2, uav_1) \vee PO(\bigcirc uav_2, uav_1)) \quad (3.26)$$

Meaning that uav_2 always follows uav_1 . In more detail, always the region occupied in the next time point by uav_2 is equal to or part of the region occupied at the current time point by uav_1 .

It is possible to use more than one next-time operator, leading to:

$$\underbrace{\bigcirc \dots \bigcirc}_k x$$

representing the state of region x k steps in the future.

Furthermore, the authors present a refinement of the continuity assumption of RCC-8:

$$\Box(EQ(x, \bigcirc x) \vee O(x, \bigcirc x))$$

meaning that region x at the next state will either remain the same or will overlap with the current region x .

3.3.2.3 ST_2

The language ST_2 further extends ST_1 by allowing application of the remaining two temporal operators, eventually \Diamond and always \Box to region variables. It is possible to express $\Diamond x$ which represents all the points that will belong to region x in the future, and $\Box x$ which represents the common points of all future states of region x .

For example, it is possible to express the following formulas:

$$P(\textit{Europe}, \Diamond EU) \tag{3.27}$$

$$\Box DC(\Diamond uav_1, \Diamond uav_2) \tag{3.28}$$

Meaning that the current region of Europe will be part of the EU at some point in the future (formula 3.27) and that the paths of the UAVs do not cross in the future (formula 3.28).

As mentioned above, $\Diamond x$ represents all the points that will belong to region x in the future, and $\Box x$ represents the common points of all future states of region x . These correspond to the union and intersection respectively of all future states of the region, as they are in fact formally defined. This presents the problem of infinite unions and intersections, which the authors limit by introducing two possible limitations, the *finite change assumption* which states “No region can change its spatial configuration infinitely often” and the *finite state assumption* which states “Every region can have only finitely many possible states (although it may change them infinitely often)”.

3.3.2.4 ST_i^+

The three ST_i^+ languages extend the corresponding ST_i languages by allowing union and intersection on region variables and region variables extended

by temporal operators. The extension of RCC-8 that allows Boolean operators is denoted BRCC-8 [Wolter and Zakharyashev, 2003].

3.4 Reasoning engines

Spatial and temporal reasoning can be treated as a constraint satisfaction problem (section 3.1). Therefore, it is possible to use CSP solvers for spatial and temporal reasoning.

There exist calculus-specific CSP solvers, such as the solver for RCC-8 by Renz and Nebel [1998], and generic CSP solvers, such as the QAT¹ (Qualitative Algebras Toolkit) project [Condotta et al., 2006a,b], SparQ² (Spatial Reasoning Done Qualitatively) [Wallgrün et al., 2007], GQR³ (Generic Qualitative Reasoner) [Gantner et al., 2008]. The distributions of all these general constraint solvers include specifications for the most commonly used qualitative spatial and temporal calculi and provide functionalities for defining new ones.

QAT [Condotta et al., 2006a,b] is a constraint programming library written in Java, which provides generic tools for describing and reasoning over qualitative calculi. It supports calculi with arbitrary arity which are specified using XML-based descriptions. Furthermore it provides different methods for qualitative constraint networks, such as the consistency problem, finding all the solutions, the minimal network problem and all these are supported by different heuristic approaches for the order of selecting variables or constraints.

SparQ [Wallgrün et al., 2007] is a constraint solver for binary and ternary calculi. It is written in Lisp and new calculi can be specified in Lisp-like syntax. It supports mapping between qualitative and quantitative information, computing relation operations and constraint-based qualitative reasoning. Furthermore, SparQ can be integrated in other applications as it can be run as a server, with communication performed over TCP/IP.

3.4.1 GQR

GQR [Gantner et al., 2008] is a generic solver for binary qualitative constraint networks. It is written in C++ and new calculi can be specified in text format or using XML-based descriptions. It uses path consistency and backtracking for solving constraint networks. GQR has been designed to be “fast and extensible generic solver” [Gantner et al., 2008]. The authors implement certain features to achieve better efficiency such as: known tractable subclasses of the calculi are used to speed up the reasoning time; weight and cardinality heuristics for selecting constraints; efficient queue data structure

¹<http://www.cril.univ-artois.fr/~saade/QAT/index.php?entry=home>

²<http://www.sfbtr8.uni-bremen.de/project/r3/sparq/>

³<http://sfbtr8.informatik.uni-freiburg.de/R4LogoSpace/Tools/gqr.html>

[Beek and Manchak, 1996] (compared to the C++ STL⁴ implementation); bit vectors are used for relations resulting in memory and speed efficient solution; caching of composition and converse results. Through performance evaluation the authors have shown that their solution is scalable, although slower than calculi specific reasoners such as the solver for RCC-8 by Renz and Nebel [1998] and the solver for Interval Algebra by Nebel [1997].

The path-consistency algorithm Gantner et al. [2008] used in GQR (algorithm 3.1) is based on the one by (ref mackworth) and it runs in $O(n^3)$ time and $O(n^2)$ memory.

Listing 3.1 Path-consistency algorithm

```

 $Q \leftarrow \{(i, j) | 1 \leq i < j \leq n\}$ 
while  $Q$  is not empty do
  select and delete  $(i, j)$  from  $Q$ 
  for  $k \leftarrow 1 \rightarrow n, k \neq i$  and  $k \neq j$  do
     $t \leftarrow C_{ik} \cap (C_{ij} \circ C_{jk})$ 
    if  $t \neq C_{ik}$  then
       $C_{ik} \leftarrow t$ 
       $C_{ki} \leftarrow t^\sim$ 
       $Q \leftarrow Q \cup \{(i, k)\}$ 
    end if
     $t \leftarrow C_{kj} \cap (C_{ki} \circ C_{ij})$ 
    if  $t \neq C_{kj}$  then
       $C_{kj} \leftarrow t$ 
       $C_{jk} \leftarrow t^\sim$ 
       $Q \leftarrow Q \cup \{(k, j)\}$ 
    end if
  end for
end while
return  $(V, C)$ 

```

For generating consistent scenarios Gantner et al. [2008] use an approach that selects different instantiations from constraints containing disjunctions of relations based on weights assigned to the base relations. Furthermore, the search is performed with an approach called *chronological backtracking* that searches through the possible instantiations of constraints and backtracks to the latest changed constraint when the search does not lead to a solution (algorithm 3.2).

⁴Standard Template Library

Listing 3.2 Scenario-consistency algorithm

```
Path-consistency( $V, C$ )
if  $C$  contains the empty relation then
  return false
end if
if there are non-basic edges then
  pick such an edge  $e = (i, j)$ 
  for all base relations  $b$  in the label of  $e$  do
     $C_{ij} \leftarrow b$ 
    if Consistent( $V, C$ ) then
      return true
    end if
  end for
end if
return false
```

Chapter 4

Analysis

In this chapter we provide an analysis of the system for spatio-temporal reasoning in ROS and describe our proposed solution for incorporating spatial reasoning into the existing formula progressor.

The chapter starts with description of the spatio-temporal stream reasoning architecture designed for usage in ROS. In the following section, the Stream Reasoner module of the architecture is discussed, mainly introducing the implementation and functionality of the formula progressor. Next, the spatial reasoning specifics are described and the extensions needed to the original formula progressor [Heintz, 2009] in order to support progression of spatio-temporal formulas.

4.1 Spatio-temporal stream reasoning in ROS

As discussed before, DyKnow is a stream processing middleware framework and can be used for reasoning over streams of data. In the execution monitoring scenario it is used to evaluate monitor formulas as information about the environment become incrementally available. The formulas that are evaluated may contain different features that are being monitored, such as altitude, speed, distance, spatial relations to other objects, etc. The values for these features are obtained through various sensors and DyKnow provides output streams for these sensory input sources.

In order to support spatio-temporal stream reasoning over incrementally available data, there are certain issues to be considered. First, it should be possible to incrementally evaluate spatio-temporal monitor formulas, which can be achieved with formula progression and spatio-temporal reasoning. Second, in order to progressively evaluate a formula the streams providing the values need to be mapped to the corresponding formula features. The mapping can be done either manually or automatically using semantic matching [Dragisic, 2011]. Third, since values in different streams may arrive at different times, the streams need to be synchronized.

The general goal is to provide ROS implementation of DyKnow. DyKnow relies on the publisher/subscriber communication model between knowledge processes and ROS provides this model via nodes and topics. Hence, knowledge processes in DyKnow correspond to nodes in ROS, and streams in DyKnow correspond to topics in ROS. Therefore, each node can subscribe and publish on multiple topics, the same as knowledge processes can have multiple input and output streams.

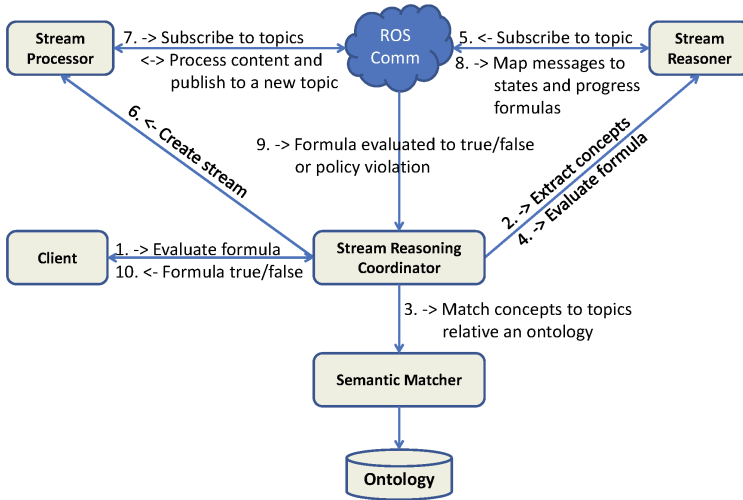


Figure 4.1: Overview of the DyKnow architecture for ROS

The design of the spatio-temporal stream reasoning architecture in ROS is presented in figure 4.1. As can be seen, the system is built around the ROS communication platform. The system consists of three main parts:

- The *Stream Reasoner* which is responsible for evaluating spatio-temporal formulas;
- The *Stream Processor* which is responsible for adapting topics to streams and preparing the streams for the stream reasoner;
- The *Semantic Matcher* which is responsible for providing automatic semantic matching between topics and features;

These three parts are managed by the *Stream Reasoning Coordinator*.

The request from the client to evaluate a formula (step 1) is handled by the Stream Reasoning Coordinator (SRC) and is then forwarded to the

Stream Reasoner which extracts the features (step 2). For the features from the formula the SRC verifies that are defined in the ontology and if there are matching topics for each feature, the stream specification that contains topic labels (fields) for the corresponding features is created (step 3). The stream specification defines a single stream that contains all the streams for the features from the formula. Next, the request to evaluate the formula is sent to the Stream Reasoner with the formulas, field to feature mapping and the name of the topic for the state stream (step 4) and then the Stream Reasoner subscribes to the received topic (step 5). The state stream specification is sent to the Stream Processor that subscribes to the topics specified in the state stream specification and creates the specified topic for the state stream (step 6). Next, the Stream Processor performs processing over the input from the topics, such as merging and synchronization and prepares the messages to be published on the state stream topic (step 7). The Stream Reasoner subscribes on the state stream topic and receives the messages, which are then mapped to states and the formulas are progressed over the states (step 8). The results of the evaluation are published on a Stream Reasoner topic which the SRC is subscribed to (step 9) and based on the result of the evaluation the SRC might take some further actions or forward the result to the client (step 10).

4.2 Stream Reasoner

The work presented in this thesis project is mainly focused on the Stream Reasoner module of the stream reasoning architecture presented above, in particular on the formula progressor. The main goal is to provide spatio-temporal reasoning capabilities. For our solution, based on literature overview we have selected the RCC-8 calculus (section 3.2.2) for the spatial reasoning part, since it is widely studied and used approach and its properties are well-known. Furthermore the PSTL logic (section 3.3.2) was selected for our solution, the language ST_0 in particular, since the logic adds a temporal dimension to RCC-8 and the syntax is similar to MTL. Therefore it is possible to express MTL formulas that include RCC-8 terms and to perform spatio-temporal reasoning based on PSTL. To perform spatio-temporal reasoning the spatial reasoning is performed independently from the temporal evaluation, hence for the spatial reasoning that is performed over the RCC-8 relations the GQR reasoner (section 3.4) is selected as it supports reasoning in RCC-8 and it is written in C++ and can be directly used in our implementation.

The formula progressor performs evaluation of spatio-temporal formulas using progression. As input data are continuously received, the values and the states of the monitored features change. Therefore the task of the formula progressor is to continuously evaluate a set of given formulas, as input data in the form of state samples that update the values of the features in the formulas incrementally become available.

The following presents a high level outline of the steps performed during formula progression:

- A set of formulas and a state-stream specification are provided as input;
- Each formula is parsed and the features are extracted and assigned types based on the state-stream specification;
- The progressor subscribes to the corresponding stream to receive updates in the form of state samples;
- As state samples arrive, the values of the features are continuously updated for each time step and the formulas are continuously progressed until there are no active formulas remaining;
- As a formula is evaluated to true or false, the formula is removed from the set of active formulas.

A detailed step-by-step description of the process performed on a concrete scenario is presented in section 5.2

The formula environment, represented by the `Environment` class, is responsible for the operation of the formula progressor. The `Environment` contains the `FormulaKeeper` which is responsible for parsing and evaluating formulas. The symbol values needed for evaluating formulas, are contained in the `Environment`. Furthermore, the `Environment` contains the spatial reasoner, represented by the `Reasoner` class which is responsible for providing the GQR reasoning capabilities to the formula progressor.

4.3 Spatial Reasoning

The spatial reasoning module is intended as an extension to the existing temporal-logic formula progression within DyKnow. The spatial reasoning module uses GQR [Gantner et al., 2008] (section 3.4) at its basis. GQR is a generic CSP solver, hence it provides functionalities for calculating path consistency and generating consistent scenarios, both of which are used in our implementation. Furthermore, GQR supports different calculi. In our implementation we are focused on RCC-8, so GQR is only used for spatial reasoning in RCC-8. However, since GQR supports other calculi, it would be possible to add further support for other calculi in our implementation as well, with some modifications.

Since we are using a region based calculus, in our implementation the regions as basic spatial entities are represented by `Region` objects. In GQR the spatial entities are represented by numbers, which are used as IDs of the regions in our implementation. In our system the regions are named, so the ID of a region corresponds to its representation in the underlying reasoner. The `SpatialRelation` class represents the relation between two

Region objects. The current possible relations are defined as the eight RCC-8 relations, but other relations for other calculi can be easily defined. The **KnowledgeBase** is a wrapper around the underlying representation in GQR of spatial entities and binary relations. In GQR, each region is represented by a number and a matrix is used for the relations between each pair of regions. Therefore, the **KnowledgeBase** class provides mechanisms for accessing and modifying the relations as well as mapping between the GQR representation of regions and the named regions used in our system. Furthermore, the **Reasoner** class provides access to the path consistency and scenario consistency methods of GQR.

There are two main issues that need to be addressed in our implementation, that is how we deal with definite and indefinite knowledge in the spatial knowledge base, and how we interpret incoming updates to the spatial knowledge base.

Indefinite knowledge for a pair of regions is represented as a disjunction (union) of spatial relations, meaning it is not known which one of those relations hold between the pair. Indefinite knowledge can occur because of uncertainty in the observations and hence in the input, or it can be the result of applying the path consistency algorithm. The issue with getting a disjunction of relations for a pair of regions that is monitored in a formula is that one cannot be certain which one of the possible relations hold. Even though the relation included in the formula is among the possible relations for the pair, the relations are JEPD so only one holds and therefore we cannot be sure if it is the one in the formula. However, if the relation included in the formula is not in the possible relations for the pair, then we can be certain that the relation does not hold and then the spatial term is false.

In our implementation, when there is only a single relation for a pair of regions for an algebraically closed network, a spatial term is evaluated to true if the relation is the same as in the formula, or false otherwise. In the case when there are multiple possible relations (disjunction) for a pair of regions, if the possible relations do not contain the relation corresponding to the spatial term the term is evaluated to false. If the possible relations contain the term, then a copy of the constraint network is made with the relation for the pair of regions set to the one in the formula and a consistent scenario is generated. If such a scenario exists, then the term is evaluated to true, otherwise to false. For example when there is a disjunction of relations, if the formula contains the spatial term $DC(uav, restricted)$ and after enforcing algebraic closure to the constraint network, the result is $uav EC, PO restricted$ then term is evaluated to false. On the other hand, if the result is $uav DC, EC restricted$, if a consistent scenario where the relation between the uav and $restricted$ is set to DC is successfully generated, the term is evaluated to true and otherwise to false. If there are multiple spatial relations that contain disjunctions of base relations, this process is performed for each spatial relation separately. This does not solve the uncertainty issue

with the result when a term or formula is evaluated to true, because it is not possible to be certain about the relation due to insufficient knowledge about the spatial environment or due to imprecise input.

The second issue, regarding the incoming updates is how the spatial relations that arrive as updates are interpreted. There are several different aspects that need to be taken into consideration, therefore we need to make certain assumptions regarding the format and interpretation of the incoming state samples. First of all we need to decide for our implementation what does a state (sample) represent with respect to the spatial knowledge base and the spatial environment. One option is to consider that the relations contained in the state sample provide full representation of the current state of the spatial environment. In this case, the spatial relations between all regions need to be included in each state sample and the spatial knowledge base is recreated with every state sample. Another option, and the one that we have taken in our implementation, is to have a spatial knowledge base loaded in advance with the initial state of the spatial environment, and the state samples contain only the relations that have changed since the previous state. This means that if there is no change in the relation between a pair of regions, there will be no relation for that pair included in the state stream. The relations that are included in the state stream replace the relations in the knowledge base. Note that with the chosen approach it is possible to also include the relations between all regions if desired, therefore providing the full state of the spatial environment instead of just the changes.

Another aspect to be considered is whether states can contain indefinite knowledge, i.e. if it is allowed to have disjunctions of spatial relations specified for a pair of regions. In the implementation, the relations in the current state replace the relations in the previous state, so then it is possible to include either definite or indefinite knowledge in the state sample. Therefore if only one relations is present for a pair of regions it is treated as definite knowledge and the new relation replaces the relations from the previous state. If multiple relations for a pair of regions are present in the state sample, it is treated as indefinite knowledge. The disjunction of spatial relations replaces the relations from the previous state.

To summarize, in the presented implementation the spatial knowledge base is loaded in advance and the spatial relations received in the state stream are considered as relations that have changed since the last state. Furthermore, only the relations that hold between regions are specified and there can be multiple relations for a pair of regions in a state stream.

4.4 Extending the MTL progressor

The original DyKnow progressor [Heintz, 2009] evaluates metric temporal logic (MTL) formulas. For our scenario, in order to provide spatio-temporal reasoning using formula progression, the progressor should be capable of evaluating spatio-temporal logic formulas, therefore further modifications of

the existing progressor are needed.

The first extension needed is to modify the formula progressor to recognize the spatio-temporal formulas, so the formula parser needs to be modified to be able to parse formulas containing RCC-8 relations. After parsing a formula, the progressor creates an *expression* which is later evaluated as updates arrive at different time steps. The expression contains other expressions which correspond to operations in the formula and *symbols* which correspond to the features in the formula. The expressions can be typed and either unary or binary. Each symbol is typed and represents the name and type of the feature. The parsing of a formula is performed by code generated by ANTLR¹ based on the rules defined for the *MTL* language. In order to parse spatio-temporal formulas the ANTLR rules need to be modified. However, we have chosen a different approach where the existing *MTL* specification is used for parsing and RCC-8 relations are treated and parsed as variable symbols like any other features with arguments would be parsed, but their type is set as a specially defined *RCC8-boolean-relation*. The types of the symbols need to be specified in advance, and this is done with the state-stream specification. For RCC-8 relations, on the other hand, the type is detected based on the name of the symbol. If the name of the symbol is one of the eight base RCC-8 relations, then the type of that symbol is set to *RCC8-boolean-relation*.

The values of the symbols are kept in a *symbol table* and are updated continuously as the new values arrive as state samples on the state stream. As these symbols are typed it is possible to perform Boolean operations on them based on their types, such as and, or, implies on Boolean symbols and greater/lower than on integers. So, depending on the type and the values of the symbols the expression can be evaluated to true or false.

For example, suppose the term $p \wedge q$ is part of a formula where both p and q are Boolean, the corresponding symbols need to be Boolean and their values are continuously updated as state samples arrive and the binary expression is continuously evaluated. For the term $x > 10$ where x is integer, the corresponding symbol is of type integer, so the operation “greater than” can be performed, i.e. the binary expression can be evaluated.

When dealing with formulas that contain RCC-8 relations, the relations are also treated as symbols but their values are obtained differently. Although it would be possible to also keep the values of the relations between pairs of regions in a symbol table, that would only be possible for relations for which we have direct observations and hence definite knowledge. However, as this might not be always possible, there is the need to perform spatial reasoning to determine the relations between pairs of regions. Therefore, instead of symbol tables, the values of RCC-8 relations are obtained through the spatial knowledge base, which provides mechanisms for access to the internal representation of regions and spatial relations of the reasoner GQR. As state updates arrive, the new relations are added to the spatial

¹Another Tool for Language Recognition - <http://www.antlr.org>

knowledge base. The new relations observed and received can have an effect on the relations between other pairs of regions and thus change the state of the constraint network. Therefore, algebraic closure should be enforced on the constraint network. We perform *lazy evaluation*, meaning that the algebraic closure of the network is only enforced when a relation needs to be retrieved from the spatial knowledge base, and new knowledge has been received and added to the knowledge base since the last time the algebraic closure of the constraint network was calculated. So, instead of using symbol tables for the values of the RCC-8 relations from the formula, their values are retrieved via calls to the external reasoner.

For example, the term $DC(x, y)$ is treated as RCC-8 relation symbol in a formula. When RCC-8 relations arrive as updates, the spatial knowledge base is updated with the received relations. Next, when the formula is evaluated, the relation between the regions x and y is retrieved from the spatial knowledge base and compared with the term $DC(x, y)$ to check if the obtained relation is DC or not.

Evaluating the spatial terms of a formula consists of ensuring that the spatial knowledge base is consistent and that the specified relations in the formula hold for region-pairs of interest by checking their value in the knowledge base. This, however, depends on how definite and indefinite knowledge is treated, as discussed in the previous section.

Chapter 5

Implementation

This chapter presents the details of the ROS implementation of the formula progressor. Some details about the design and implementation of the progressor are also available in sections 4.3 and 4.4. The implementation is described in detail and a specific scenario is presented to highlight each of the phases of the formula evaluation process.

5.1 ROS implementation

The ROS implementation of the progressor is part of the spatio-temporal stream reasoning architecture described in section 4.1. Furthermore, as already described, the system is designed to be modular so each part can be used independently.

The formula progressor is independent and it can be used as a separate unit by itself. The implementation provides an importer interface for handling continuously arriving data in the form of state samples in a state stream, and an interface for publishing the results of an evaluation. Therefore, to be used on different systems, specific implementations of the importer and publisher interfaces need to be provided.

For our scenario, we require a ROS implementation of the formula progressor, therefore it is implemented as a ROS node. The node publishes a service for evaluating formulas. The goal is the progressor to be run as a node and to be able to receive requests for evaluating multiple formulas during run-time. Some of the formulas may be related to others, either only in a logical sense or may be defined over the same domain for example. Therefore, it should be possible to group related formulas and perform evaluation on a group of formulas, and also to evaluate multiple groups of formulas. The main operation of the node is performed by the `ROSProgressorManager` which creates and manages instances of `ROSProgressor`. The `ROSProgressor` is responsible for performing evaluation on a group of formulas, which receive updates on the same state-

stream. This involves managing the formula evaluation environment including the spatial reasoner, the state-stream importer and the result publisher. As the progressor as a ROS node is capable of performing evaluation on multiple groups of formulas subscribed on multiple different topics, the `ROSProgressorManager` manages multiple instances of the `ROSProgressor`, each of which performs evaluation on its own group of formulas. Furthermore, the `ROSProgressorManager` is responsible for subscribing to the input topics and forwarding the updates received to the interested `ROSProgressor` instances. It is also responsible for advertising the topics and publishing the results of the evaluation as they become available.

The stream reasoner ROS node advertises a service for evaluating formulas, namely the service `evaluate_formulas`:

Listing 5.1: EvaluateFormulas.srv

```
string [] formulas
string state_stream_topic
dyknow_msgs/Domain [] domains
dyknow_msgs/Region [] regions
dyknow_msgs/RCC8Relation [] initial_relations
dyknow_msgs/FieldFeatureType [] field_to_feature_mapping
-----
bool success
string error_message
int32 formula_group_id
int32 [] formula_ids
string formula_result_topic
```

As it can be seen in the listing above, the service request contains an array of formula strings, the name of the topic where the updates are published (the topic to subscribe to), the contents of the domains in the formula, if any, arrays of regions and spatial relations representing the spatial environment if the formulas contain spatial terms, and the field-to-feature mapping. The response contains the success value, an error string, the ID of this formula group, the IDs of all the formulas in this formula group and the name of the topic on which the results will be published when the formulas are evaluated.

The domain specification in the request is provided by an array of `Domain` messages, which contain the name of the domain, and an array of strings representing the objects in the domain.

Listing 5.2: Domain.msg

```
string domain
string [] objects
```

Next, the request may contain an array of regions to be added and an array of initial spatial relations. The regions defined in the `Region` have a name which is the representation of the region in the formulas. In addition

a region may contain a string that represents the type of the region and a Boolean field that specifies whether the region is dynamic or static, both of which are mainly intended for future use. Each spatial relation, as defined in the `RCC8Relation` message contains the names of the two regions and a string array corresponding to the RCC-8 relations between the pair.

Listing 5.3: Region.msg

```
string name
string type
bool static
```

Listing 5.4: RCC8Relation.msg

```
string region1
string region2
string [] rcc8_relation
```

The request contains the field-to-feature mapping which is an array of string triplets, field, feature and type, represented by the `FieldFeatureType` message. The mapping is between fields in the state stream and the corresponding features in the formulas. The type is needed as symbols in the formula progressor are typed, and the symbols are created during the parsing of the formula. For example, for the simple formula `always [0, 1000] altitude[uav1] > 10`, the mapping could be the triplet (`‘‘a-1’’`, `‘‘altitude[uav1]’’`, `‘‘double’’`). This means that in the state samples received, the string `a-1` corresponds to the feature `altitude[uav1]` in the formula and the values received are of type `double`.

Listing 5.5: FieldFeatureType.msg

```
string field
string feature
string type
```

As soon as the server receives the request to evaluate formulas and the request is processed, the progressor (service) subscribes to the topic for updates, and advertises a topic where the results will be published. In the response message, the service sends the name of this topic. The client is expected to subscribe to the topic and wait for the results. In addition, in the response message the ID of the formula group and the IDs of all the formulas are included.

The updates on the topic (which represents a state stream) arrive in the form of a timed state samples, represented by the `Sample` message. In addition to the ROS header, it contains the time for which this sample is valid and an array of fields. Each field in the array, as defined in the `Field` message, contains the name of the field and its value at the specified time point. For the previous example, we have the field `a-1` representing the feature `altitude[uav1]`, so a sample would contain a time point and the

field, which in fact will have the name `a-1` and some floating point value. The values are received as strings, however they are converted to the correct type for the evaluation of formulas.

Listing 5.6: Sample.msg

```
Header header
time valid_time
Field [] fields
```

Listing 5.7: Field.msg

```
string name
string value
```

As the updates are received continuously, the formulas get eventually evaluated to either true or false. As mentioned before, the client is responsible to subscribe on the topic published by the progressor service and wait for the result of the progression. The result is in the form of `FormulaResult` message which contains the success of the operation (not to be confused with the result), the error message, the result of the evaluation of the specified formula, the string representation of the formula, the valid time, the ID of the formula and the ID of the formula group in which the formula belongs to. Therefore, if there are multiple formulas in the group, there will be one message of this kind for each formula.

Listing 5.8: FormulaResult.msg

```
bool success
string error_msg
bool result
string formula
time valid_time
int32 formula_id
int32 formula_group_id
```

5.2 Scenario

In this section we present a concrete use-case scenario, where a specified formula is being evaluated and the whole process is explained in detail.

The following spatio-temporal formula is used:

$$\forall u \in UAV \square_{[0,600000]} [altitude(u) > 20 \wedge DC(u, restricted)] \quad (5.1)$$

The formula is interpreted as “the next ten minutes all UAVs must have altitude higher than 20 meters and not fly over the restricted area”. The intervals in the formulas are specified in milliseconds. Furthermore for this

example scenario we assume a sampling period of 100 milliseconds. For simplicity we only show the first samples below.

Since the formula is spatio-temporal, there needs to be a spatial knowledge base present so spatial reasoning can be performed. Assume we have an environment consisting of a road, a property on the side of the road, a building inside the property, a forest on the other side of the road and a restricted region in the forest. In addition, we have two UAVs located above the road (figure 5.1).

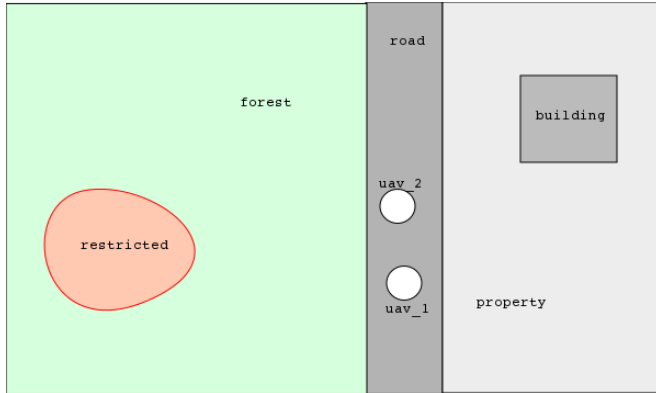


Figure 5.1: Scenario environment

As discussed before, the spatial knowledge base is loaded in advance by a call to the `load_spatial_kb` service. The request sent to the service contains the ID of the group of formulas, for example the ID is 1, and contains the following regions and RCC-8 relations:

- Region 1:** name = road ;
- Region 2:** name = property ;
- Region 3:** name = building ;
- Region 4:** name = forest ;
- Region 5:** name = restricted ;
- Region 6:** name = uav1 ;
- Region 7:** name = uav2 ;

RCC-8 relations:

region1	region2	rcc8relation
road	property	EC
road	building	DC
road	forest	EC
road	restricted	DC
road	uav1	NTPPi

road	uav2	NTPPi
property	building	NTPPi
property	forest	DC
property	restricted	DC
building	forest	DC
building	restricted	DC
building	uav1	DC
building	uav2	DC
forest	restricted	NTPPi
forest	uav1	DC
forest	uav2	DC
restricted	uav1	DC
restricted	uav2	DC
uav1	uav2	DC

Note that, however, not all relations between all pairs of regions need to be specified as some of the relations can be inferred from the specified relations by the reasoner. For example, if we have $DC[property, forest]$ and $NTPP[building, property]$ then it can be inferred that $DC[building, forest]$. However it is preferred to have the initial knowledge base as specified as possible since it reduces the indefinite knowledge. Furthermore, as each of the RCC-8 relations have inverse relations, the ordering of the pair of regions is not important.

In order to evaluate the formula specified above (5.1), a request to the `evaluate_formulas` service needs to be sent. The request would contain the following:

Listing 5.9: `evaluate_formulas` service request

```

formulas = {
  Formula 1:
    "forall u in UAV always [0, 600000]
      (altitude[u] > 20 and DC[u, restricted])"
}
state_stream_topic = "state-stream-1"
domains = {
  Domain 1:
    name = "UAV"
    objects = {"uav1", "uav2"}
}
regions = {
  Region 1:
    name = "road"

```

```
Region 2:
  name = "property"
Region 3:
  name = "building"
Region 4:
  name = "forest"
Region 5:
  name = "restricted"
Region 6:
  name = "uav1"
Region 7:
  name = "uav2"
}
initial_relations = {
  RCC8Relation 1:
    region1 = "road"
    region2 = "property"
    rcc8relation = EC
  RCC8Relation 2:
    region1 = "road"
    region2 = "building"
    rcc8relation = DC
  RCC8Relation 3:
    region1 = "road"
    region2 = "forest"
    rcc8relation = EC
  RCC8Relation 4:
    region1 = "road"
    region2 = "restricted"
    rcc8relation = DC
  RCC8Relation 5:
    region1 = "road"
    region2 = "uav1"
    rcc8relation = NTPPI
  RCC8Relation 6:
    region1 = "road"
    region2 = "uav2"
    rcc8relation = NTPPI
  RCC8Relation 7:
    region1 = "property"
    region2 = "building"
    rcc8relation = NTPPI
  RCC8Relation 8:
    region1 = "property"
    region2 = "forest"
```

```
    rcc8relation = DC
RCC8Relation 9:
    region1 = "property"
    region2 = "restricted"
    rcc8relation = DC
RCC8Relation 10:
    region1 = "property"
    region2 = "uav1"
    rcc8relation = DC
RCC8Relation 11:
    region1 = "property"
    region2 = "uav2"
    rcc8relation = DC
RCC8Relation 12:
    region1 = "building"
    region2 = "forest"
    rcc8relation = DC
RCC8Relation 13:
    region1 = "building"
    region2 = "restricted"
    rcc8relation = DC
RCC8Relation 14:
    region1 = "building"
    region2 = "uav1"
    rcc8relation = DC
RCC8Relation 15:
    region1 = "building"
    region2 = "uav2"
    rcc8relation = DC
RCC8Relation 16:
    region1 = "forest"
    region2 = "restricted"
    rcc8relation = NTPPI
RCC8Relation 17:
    region1 = "forest"
    region2 = "uav1"
    rcc8relation = DC
RCC8Relation 18:
    region1 = "forest"
    region2 = "uav2"
    rcc8relation = DC
RCC8Relation 19:
    region1 = "restricted"
    region2 = "uav1"
    rcc8relation = DC
```

```

RCC8Relation 20:
  region1 = "restricted"
  region2 = "uav2"
  rcc8relation = DC
RCC8Relation 21:
  region1 = "uav1"
  region2 = "uav2"
  rcc8relation = DC
}
field_to_feature_mapping = {
  FieldFeatureType 1:
    field = "a-1"
    feature = "altitude [uav1]"
    type = "double"
  FieldFeatureType 2:
    field = "a-2"
    feature = "altitude [uav2]"
    type = "double"
}

```

When the above `evaluate_formula` request is received by the service (figure 5.2) the `ROSProgressorManager` initializes the `ROSProgressor` for that formula group, adds the spatial regions and relations if present, the domain, the field-to-feature mapping and the formulas to the `ROSProgressor`. The `ROSProgressorManager` subscribes to the topic `state-stream-1` and advertises a topic where the results for this formula group will be published, `results-stream-1` which is returned to the client in the service response message. The `ROSProgressor` on the other hand, loads up the spatial knowledge base (figure 5.3), sets up the formula environment, the domains and the mapping and parses the formula.

After the `ROSProgressor` has been initialized, it loads the spatial knowledge base if requested by the `ROSProgressorManager`. The `ROSProgressor`, initializes the spatial reasoner and the knowledge base, loads up the regions and relations and enforces algebraic closure on the constraint network (figure 5.3). Note that if no spatial terms are used in the formula, there is no need to load and use the spatial reasoner and the knowledge base.

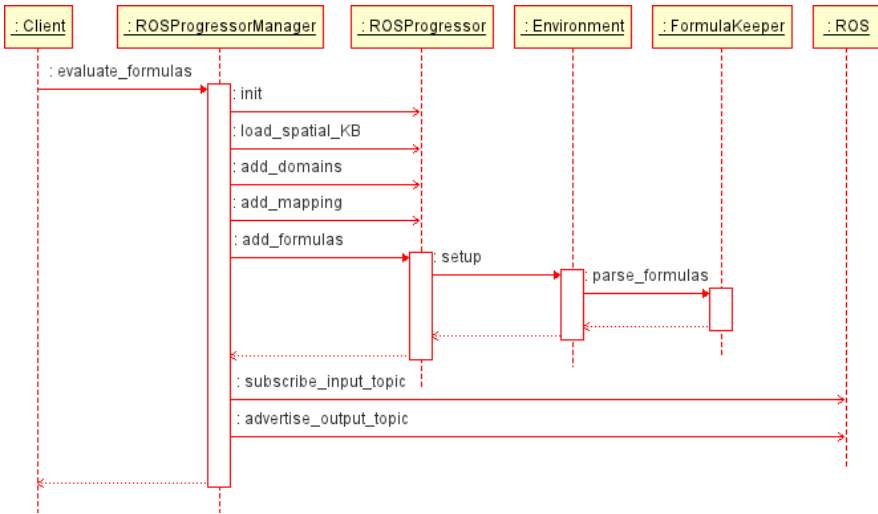
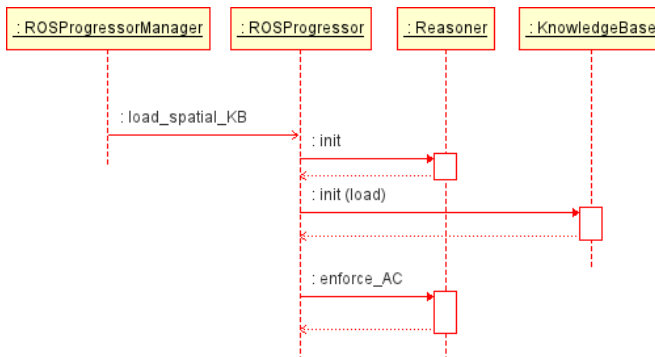
After the formula has been parsed, the result is an *expression* which has the following string representation:

```

forall #q-u-0
  (always [0, 600000] (
    (altitude[#q-u-0 quantifier] binary_op 20) and
    (DC[#q-u-0 quantifier, restricted])))

```

which is based on the following abstract syntax tree constructed during parsing:

Figure 5.2: Sequence diagram for `evaluate_formulas` service requestFigure 5.3: Sequence diagram for `load_spatial_kb` service request

```
( forall ( QUANTIFIER_SPEC u ( DESCRIPTOR UAV ) )
  ( always ( TIME.INTERVAL 0 600000 )
    ( and
      ( > ( VARIABLE.REFERENCE
          altitude ( DESCRIPTOR u ) ) 20 )
      ( VARIABLE.REFERENCE
        DC ( DESCRIPTOR u ) ( DESCRIPTOR restricted ) )
    )
  )
)
```

As it can be seen in the abstract syntax tree, `DC` is treated the same as `altitude` and both are parsed as a `VARIABLE.REFERENCE`. When the symbols

are created, the type of the symbol representing the feature *altitude* is set to `double` based on the feature-to-field mapping provided, and the type of the symbol representing *DC* is set to `rcc8.boolean_relation` since the name of the symbol is one of the base RCC-8 relations.

In the produced expression the arguments of *altitude* and *DC* are tied to the quantifier *u*, namely `#q_u_0`. The variable symbols, however, are expanded based on the contents of the domain the quantifier is connected to, so after the expansion we get `altitude[uav1]` and `altitude[uav2]` as well as `DC[uav1,restricted]` and `DC[uav2,restricted]`. The expanded formula looks like this:

```
(always [0, 600000]
  ((altitude[uav1] binary_op 20) and
   (DC[uav1, restricted])))
and
(always [0, 600000]
  ((altitude[uav2] binary_op 20) and
   (DC[uav2, restricted])))
```

The formula is now parsed and the expression is created, so the progression can start as soon as state samples arrive on the state stream. The `ROSProgessorManager` is subscribed to the topic and as state samples arrive, it forwards the messages to all instances of `ROSProgessor` that are waiting on updates on that topic, since multiple formula groups can receive updates on the same topic. The state samples arrive as a `Sample` message. For example, let us assume that initially both UAVs are at altitude of 25 meters and the spatial relations remain the same as specified within the spatial knowledge base. The following is an example of the first `Sample` message that arrives:

Listing 5.10: Example state sample message

```
Sample:
  header:
    ...
  valid_time:
    secs = 0
    nsecs = 0
  fields:
    field 1:
      name = "a-1"
      value = "25.0"
    field 2:
      name = "a-2"
      value = "25.0"
```

Note that for simplicity in the examples the header is not included in the listing, and we assume the valid time starts at 0, even though the system

will internalize the time and start the progression from the time set in the first sample received.

As the `ROSProgressorManager` receives the message (figure 5.4) it forwards it to the `ROSProgressor` and the update is handled by its importer. The importer then converts the state sample time to the internal time representation of the progression and then starts the progression of the formula over the newly received state. Actually when the first state sample is received the initial symbol values are set. For each next state sample, the formula is progressed with the previous symbol values to a time point right before the time specified in the state sample, and the symbol values are updated to the new values contained in the new state sample. This is repeated as long as state samples are received, or the formula is evaluated to true or false.

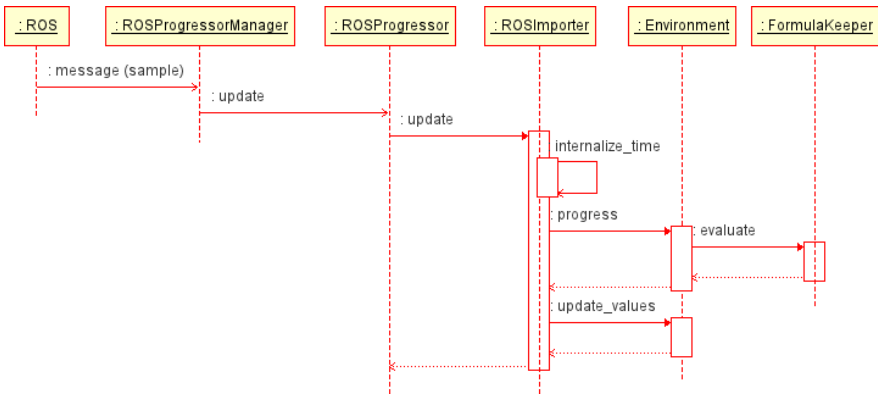


Figure 5.4: Sequence diagram for handling updates

After the first state sample message, specified above, is received, the values for the altitude of both UAVs are set. However, as no spatial relations are received, no interaction with the spatial reasoner is required in this step. Assume that in the next state, *uav1* remains at the same altitude while *uav2* rises to 30 meters and furthermore, both start moving, *uav1* towards the forest and *uav2* towards the property and they both move to the edge of the road. The following is the corresponding `Sample` message:

Sample :

```

header :
  ...
valid_time :
  secs = 0
  nsecs = 100000000
fields :
  field 1 :
    name = "a-1"
  
```



```

        value = "25.0"
    field 2:
        name = "a-2"
        value = "30.0"
    field 3:
        name = "TPP(uav1, road)"
        value = ""
    field 4:
        name = "EC(uav1, forest)"
        value = ""
    field 5:
        name = "TPP(uav2, road)"
        value = ""
    field 6:
        name = "EC(uav2, property)"
        value = ""

```

From the example state sample it can be seen that the non-spatial features in the formula are represented by the corresponding fields. The spatial relations that have changed are also included in the state sample and they are represented by a separate field for each new relation.

When this state is received, the time is internalized to 100 and the formula is progressed with the values of the previous state to time point 99 and then the values of the newly received state sample are set. The altitudes for both UAVs have been set in the initial state to 25 meters, and when evaluating the formula those values are obtained from the symbol table. For the spatial part of the formula however, the values are obtained via a call to the spatial reasoner to retrieve the relation between the pairs (*uav1, restricted*) and (*uav2, restricted*) and checked if both pairs are *DC*. Since there has been no change in the spatial relations, the formula is evaluated to true after the progression and the progression is continued with the following expression:

```

(always [0, 599900]
  ((altitude[uav1] binary_op 20) and
   (DC[uav1, restricted])))
and
(always [0, 599900]
  ((altitude[uav2] binary_op 20) and
   (DC[uav2, restricted])))

```

This expression is identical to the one produced at the start of the progression except for the interval. The interval indicates that this expression should be true for the next 599900 milliseconds. Such expressions for smaller time intervals are produced after each progression over a state sample until the formula is progressed over the whole remaining time interval or is evaluated to false.

After the formula has been progressed to time point 99, the symbol values are updated with the new values received in the new state sample. In this second state sample, the spatial relations that are received are added to the spatial knowledge base.

The next state sample, listed below, represents a state where *uav2* rises further to 35 meters and *uav1* partially overlaps the road and the forest and *uav2* partially overlaps the road and the property.

Sample :

```

header :
    ...
valid_time :
    secs = 0
    nsecs = 200000000
fields :
    field 1:
        name = "a-1"
        value = "25.0"
    field 2:
        name = "a-2"
        value = "35.0"
    field 3:
        name = "PO(uav1, road)"
        value = ""
    field 4:
        name = "PO(uav1, forest)"
        value = ""
    field 5:
        name = "PO(uav2, road)"
        value = ""
    field 6:
        name = "PO(uav2, property)"
        value = ""

```

The same as before, the time of this state sample is internalized to 200 and the formula progressed over the previous state to time point 199. When the formula is evaluated during progression the relations between the pairs (*uav1, restricted*) and (*uav2, restricted*) need to be evaluated in the formula. However, in this case the spatial relations have changed in the previous state, so as the relations are obtained from the spatial reasoner the algebraic closure of the constraint network needs to be recalculated to reflect the changes in the spatial relations from the previous state and ensure consistency of the network. As both relations remain *DC* the formula is evaluated to true and the progression continues.

Note that no direct information was added during the previous state about the relations between the UAVs and the restricted region, only the relations between the UAVs and the road, forest and property. However as

the relations between the road, forest and property and the restricted region are specified, the reasoner can infer that the UAVs are disconnected from the restricted region. In this case as the relations between the UAVs and the restricted are specified before, the algorithm just checks the consistency of the relations. If there is no information about these relations, the reasoner will however infer this from the other specified relations between the road, forest and property and the restricted region. As discussed before, the evaluation of RCC-8 relation symbols might result to false if the relation between the pair is not the one expected in the formula or if the constraint network is inconsistent. A constraint network can become inconsistent if for example a faulty relation has been inserted in the knowledge base resulting in a state of the spatial environment that is not possible with respect to the relations in the knowledge base.

Assuming for the remainder of the execution the next state samples received satisfy the rules for the UAVs flying above 20 meters and away from the restricted region, the symbols are evaluated to true and the formula will finally be evaluated to true.

However, if any of the symbols are evaluated to false, the expression and hence the formula will be evaluated to false. For example, if at any state sample the value for the altitude is lower than 20 meters then the corresponding altitude symbol is evaluated to false and hence the full formula. Regarding the RCC-8 relation symbol in the expression, if for example at some state $EC[uav1, restricted]$ or any other relation other than DC is received then the result of the evaluation is false. Furthermore, if some newly received spatial relation leads to inconsistent spatial knowledge base the result of the evaluation is also false. Such an example is, if both $TPP[uav2, property]$ and $TPP[uav2, forest]$ are received will lead to inconsistency, since $DC[property, forest]$ is already in the knowledge base.

When the progression is over and the formula is finally evaluated to true or false, the `ROSProgessorManager` publishes `EvaluateFormulaResult` message on the topic `result-stream-1` containing the string representation of the formula and the result of the evaluation. If there are multiple formulas, one such message is sent after each formula is evaluated.

Chapter 6

Performance Evaluation

This chapter presents a performance evaluation of the formula progressor. The goal is to evaluate the progression time of a state over one or more formulas. Since there are different aspects that the progression time depends on, there are different test cases presented and evaluated, developed to highlight the effect of different properties of the formulas being evaluated and taking into consideration both temporal logic formulas and spatio-temporal logic formulas.

First the choice of test cases is presented and motivated. Next, the results are presented and discussed.

6.1 Test cases

The focus of the performance evaluation is on the execution time of the progressor. In particular, the average time it takes to progress a state over the formulas being evaluated. We are mainly interested in three different aspects of the progressors and their effect on the performance:

- Number of formulas - since the progressor can contain more than one formula, possibly related to each other, that are progressed all together when state samples arrive, we are interested how the progressor behaves for different number of formulas.
- Number of terms in a formula - usually formulas do not have a large number of terms. However, the progressor supports quantifiers and deals with quantified formulas by expanding the formulas for all the elements in the domain. This can lead to evaluating formulas with large number of terms depending on the size of the domain, which naturally influences the time it takes to progress the formula.
- Spatial reasoning - when dealing with spatio-temporal formulas, the evaluation of the spatial terms in the formula is executed through calls

to the external spatial reasoner. Therefore, we are interested in the performance characteristics of the progression when using the external reasoner and mainly the effects the size of the spatial knowledge base has on the performance.

To evaluate these aspects of the formula progression, we have defined four test cases:

Test case 1 Evaluation of the performance of the formula progressor when dealing with different numbers of temporal logic formulas. In this test case the evaluated formulas do not contain spatial and spatio-temporal terms.

Test case 2 Evaluation of the performance when dealing with different number of terms in a formula. Both temporal and spatio-temporal formulas are tested and compared.

Test case 3.1 Evaluation of the performance when different number of new spatial relations are added in the knowledge base, which requires enforcing algebraic closure of the constraint network.

Test case 3.2 Evaluation of the performance for different sizes of the spatial knowledge base, namely different numbers of regions included in the knowledge base.

In all the test cases the average time to progress a state is tested. Therefore, in each of the test cases, each formula was progressed over 1000 states (time steps) to get the average time to progress a state, and each such run was repeated 10 times. Since the focus is on the time of the progression, the sampling is simulated and the state samples are read from a file. In the experiments where spatial terms are used, the spatial knowledge base is loaded from a file before the simulation is started.

The experiments were run on a Acer Aspire 6930G with Intel Core2Duo P7350 processor at 2.00 GHz, 4 GB of RAM running OpenSUSE 11.4 with kernel 2.6.37.6-0.9-desktop x86_64. The tested implementation uses GQR release 1298.

The results of the experiments are presented in the following section.

6.2 Test results

Test case 1

This experiment is the same as the one performed by Heintz [2009] with the original formula progressor. However, since changes have been made, the goal is to test how the new implementation of the progressor handles different numbers of formulas for evaluation.

In this test case the formula used is:

Table 6.1: Average time to progress a state over different numbers of temporal logic formulas

time (ms)	p = T	p = (F, T)	p = (10 * F, T)
500 formulas	4.3	5.7	6.9
1000 formulas	8.8	11.6	14.6
1500 formulas	13.3	18.3	21.5
2000 formulas	18.0	24.2	29.0
2500 formulas	22.9	30.7	37.1
3000 formulas	27.6	37.2	44.8
3500 formulas	32.6	43.1	51.8
4000 formulas	36.9	50.5	59.1
4500 formulas	41.6	57.8	67.0
5000 formulas	45.9	65.1	74.5

$$\Box \diamond_{[0,1000]} p$$

The interpretation of the formula is that p must never be false for more than one second [Heintz, 2009]. Since the sampling time is 100 milliseconds p must become true within 10 state samples.

Furthermore, the experiment is performed starting from 10 up to 5000 instances of this formula.

The experiment is performed, as the original, on three different state sequences in order to test different progression behaviors. The input sequences used by Heintz [2009] are:

- p is always true - this is the best case scenario, as the formula is evaluated to true immediately.
- p alternates between true and false - in this case the formula is progressed when false is received and evaluated to true when true is received.
- p is false for 10 time steps and then true for 1 time step - the formula is progressed for 10 time steps as finally true is received and the formula is evaluated to true.

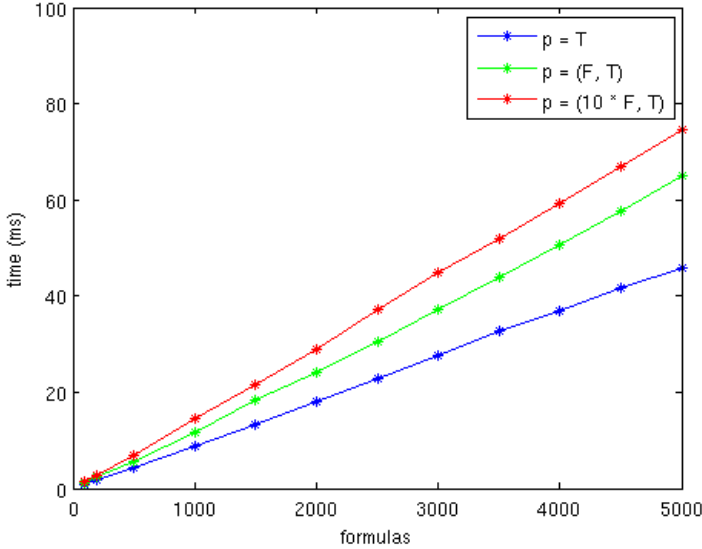


Figure 6.1: Average time to progress a state over different numbers of temporal logic formulas

The results presented in table 6.1 and figure 6.1 show that the increase of processing time to evaluate the formulas is linear with the increase of number of formulas being evaluated. The progression time varies depending on the input sequence, as expected. However, even in the worst case scenario, 5000 instances of the formula can be progressed in less than 80 milliseconds, which shows that the progressor is fast and efficient.

Test case 2

With this experiment the goal is to test the effect the number of terms in a formula have to the progression time of the full formula, since progressing a formula to the next time point entails progressing each term to the next time point.

In this test case two formulas are used, one temporal logic formula and one spatio-temporal logic formula:

$$\begin{aligned} & \Box p_1 \wedge \dots \wedge p_n \\ & \Box DC[r_1, r_2] \wedge \dots \wedge DC[r_i, r_j] \end{aligned}$$

The number of terms n is varied from 10 to 200. In the case of the spatio-temporal formula i and j are varied so the total number of terms will vary from 10 to 200. Although it would be hard to imagine that a manually

Table 6.2: Average time to progress a state over a single formula with varying number of spatial and non-spatial terms

time (ms)	temporal	spatial
10 terms	0.06	0.12
20 terms	0.12	0.24
50 terms	0.29	0.57
100 terms	0.61	1.12
150 terms	0.95	1.63
200 terms	1.29	2.09

entered formula would contain a large number of terms, formulas containing quantifiers are expanded so the number of terms might get large depending on the size of the domains.

For this experiment the formulas need to be evaluated to true, therefore formulas containing *always* and *conjunction* are used because those formulas represent the worst case scenario. When evaluating formulas and terms under the temporal operator “always” (\square) the progression needs to be performed for every time step. The conjunctions are used since all the terms in the formula need to be evaluated to true, as opposed to when disjunctions are used and only one term needs to be true for the formula to be evaluated to true.

We test both temporal and spatio-temporal formulas, as evaluating spatial terms requires calls to the external reasoner to obtain the relations, while the values of other non-spatial terms are kept in a symbol table.

The results of this experiment in table 6.2 and figure 6.2 present the average time needed to progress a state over a formula with varying number of terms. The time needed to evaluate a formula linearly increases with the number of terms in the formula. Progressing over formulas that contain spatial relations takes slightly more time, since it requires obtaining the values from the spatial reasoner. However, the time to progress formulas with large number of terms is very low, namely around 1.4 milliseconds for a formula containing 200 non-spatial terms and around 2 milliseconds for a formula containing 200 spatial terms.

Test case 3.1

In this experiment we focus on the number of new spatial relations being added to the knowledge base, and their effect on the performance. In the previous experiment for the spatio-temporal formula, no new relations are added to the knowledge base during the progression. This means that there is no need to enforce algebraic closure of the constraint network, as the relations are not changing during execution and the constraint network is

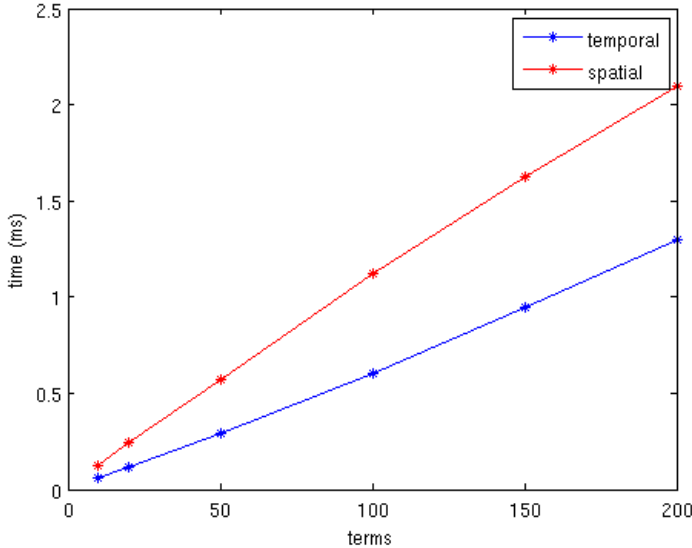


Figure 6.2: Average time to progress a state over a single formula with varying number of spatial and non-spatial terms

already consistent. Each call to the reasoner only retrieves the spatial relation between the regions. Adding new relations to the knowledge base would require enforcing algebraic closure of the constraint network so the new relations could be taken into consideration. As discussed earlier (chapter 5), we perform lazy evaluation, meaning the enforcing of algebraic closure is triggered when a relation is obtained from the knowledge base and new knowledge has been added since the last time algebraic closure was enforced.

For this experiment a formula containing 100 spatial terms is used. The experiment is performed on two different knowledge bases, one with 25 and the other with 50 regions in the knowledge base. For both knowledge bases, the number of new relations added at a time point during progression is varied from 5 to 100.

From the results presented in table 6.3 and figure 6.3 it can be seen that the number of new spatial relations being added to the spatial knowledge base at each time step of the progression, has a negligible influence on the progression time. Average progression time of a state over formulas when using knowledge base with 25 regions is around 8 milliseconds despite the variation in the number of new formulas, while the average time when a knowledge base with 50 regions is used is around 40 milliseconds for all the different tests with different numbers of new relations being added. It is obvious that the number of regions in the knowledge base is more important. This is expected as the time complexity of the path-consistency algorithm is

Table 6.3: Average time to progress a state with different numbers of regions in the KB

time (ms)	25 regions	50 regions
5 relations	6.64	41.75
10 relations	6.66	41.79
20 relations	6.70	41.83
50 relations	6.81	42.13
100 relations	6.91	42.45

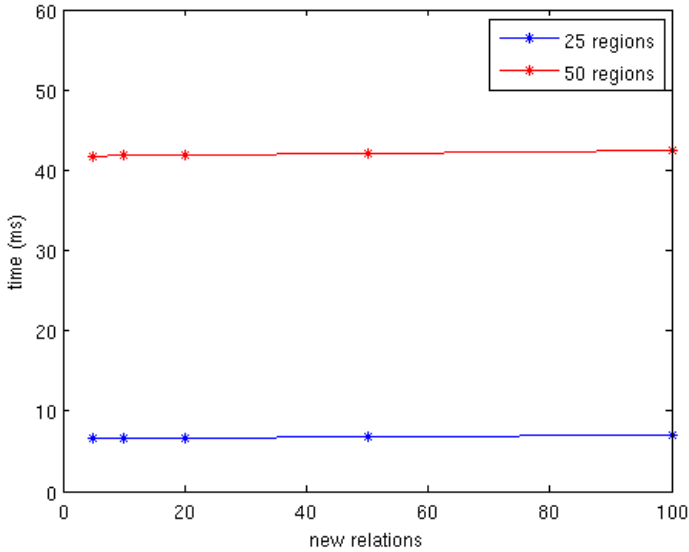


Figure 6.3: Average time to progress a state with different numbers of regions in the KB

dependent on the number of nodes in the constraint network. In our implementation we use lazy evaluation hence algebraic closure of the constraint network is enforced when the relation for a pair of regions is obtained but only if new relations have been added since the previous time the algebraic closure procedure was performed. Therefore the number of new relations do not influence significantly the progression time.

Test case 3.2

From the previous experiment it is obvious that the number of regions in the spatial knowledge base have significant influence on the performance of the progression when dealing with spatio-temporal formulas. Therefore, in this experiment the focus is on the number of regions in the knowledge base.

For this experiment a formula containing 100 spatial terms is used. At each time step in the progression, the state sample contains 10 new spatial relations which are added in the knowledge base. The experiment is performed for knowledge bases containing from 20 to 100 regions.

Table 6.4: Average time to progress a state when algebraic closure needs to be enforced

	20 regions	40 regions	60 regions	80 regions	100 regions
time (ms)	2.5	22.4	70.6	164.4	333.2

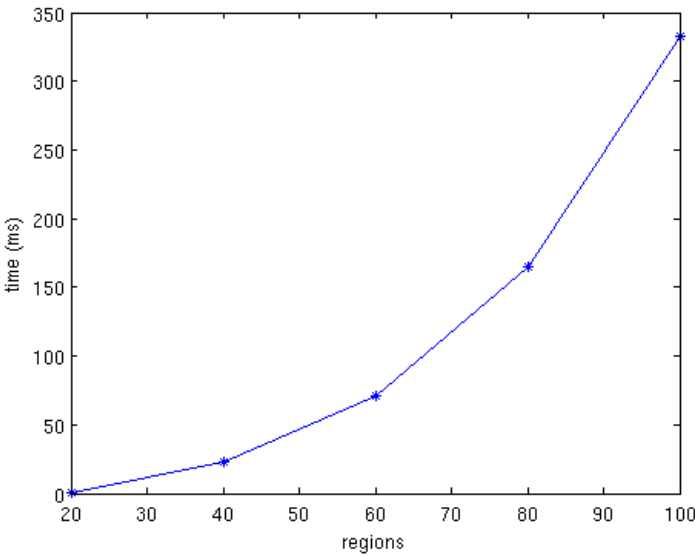


Figure 6.4: Average time to progress a state when algebraic closure needs to be enforced

Table 6.4 and figure 6.4 show the impact of the size of the knowledge base on the average time to progress a state. The average progression is fast for smaller knowledge bases but it goes up to 333 milliseconds for a knowledge base with 100 regions. Adding new spatial relations to the knowledge

base requires enforcing algebraic closure to the constraint network, therefore the size of the knowledge base has significant influence when adding new spatial knowledge, as the number of regions are the number of nodes in the constraint network and has a direct effect on the execution time of the path consistency algorithm. The results are as expected, since the path consistency algorithm has a cubic time complexity ($O(n^3)$) on the number of nodes in the constraint network.

There are certain changes specific for our scenarios that can be made to the implementation of the spatial reasoner, mainly regarding enforcing algebraic closure to a constraint network, that can improve the execution time of the path consistency algorithm. Some of these approaches are discussed in more detail in section 7.2. One of the approaches proposed in section 7.2, for using dynamic and static regions in a spatial knowledge base has been implemented and tested. In particular, the same experiment was defined for a knowledge base of 100 regions, with 95 defined as static and 5 as dynamic. The average time for progressing a state in that case is 41 milliseconds, which is considerably faster than the 333 milliseconds needed for a spatial knowledge base of 100 regions with no distinctions between the regions. However this approach requires further analysis and more thorough testing and has been left as a future work.

6.3 Discussion

The experiments presented in this chapter evaluate the performance of the formula progressor based on different aspects that the progression time depends on. The experiments produced the expected results. Three different aspects were taken into consideration and four different test cases were evaluated. The first aspect considered the number of formulas being evaluated since the progressor can evaluate multiple formulas at the same time. This experiment produced the expected results as the progression time increased linearly with the number of formulas. Furthermore it can be concluded that the progressor is fast when dealing with multiple formulas as a state was progressed over 5000 formulas in less than 80 milliseconds in the worst case scenario. The second aspect considered the number of terms in the formula, as a formula containing quantifiers is expanded during evaluation and the expanded formula can contain large number of terms. In this experiment both formulas with and without spatial terms were tested. The progression time increased linearly with the number of terms, and as expected it was a little slower for formulas containing spatial terms. The last aspect was related to the spatial reasoning and the progression time when spatial reasoning was performed during evaluation of the formulas. Two different experiments were performed to test the effect of the number of new spatial relations received at each time point and the size of the spatial knowledge base. The experiments showed that the execution time is only influenced by the size of the spatial knowledge base, namely the number of regions.

Overall it can be concluded that the formula progressor is fast and efficient when dealing with large number of formulas and formulas with large number of terms. Furthermore it is scalable, since adding more formulas or terms linearly increases the progression time. Although the progression time might be longer when updating a spatial knowledge base with large number of regions, the results conform with the time complexity of the path consistency algorithm.

Chapter 7

Conclusion

7.1 Summary

Autonomous systems require a lot of information about the environment in which they operate in order to perform different high-level tasks. The information is made available through various sources, such as remote and on-board sensors, databases, GIS, the Internet, etc. The sensory input especially is incrementally available to the systems and can be represented as streams. However, performing the functionalities of the system often require some sort of reasoning over the input data, which is often not suitable for the higher level representations needed for reasoning. DyKnow is a stream processing framework that provides functionalities to represent knowledge needed for reasoning from streaming inputs.

DyKnow has been used within a platform for task planning and execution monitoring for UAVs. The execution monitoring is performed using formula progression with monitor rules specified as temporal logic formulas. The goal of this thesis project was to extend the formula progression with spatial reasoning. For this, an overview of existing spatial and spatio-temporal reasoning theories and approaches available in the literature was done, with the main focus on RCC-8 and PSTL. Furthermore, different reasoning engines were analyzed and GQR was selected for our solution. In addition to the spatial reasoning extension, the second goal is to implement the formula progressor as a part of a spatio-temporal stream reasoning architecture in ROS.

In this thesis report we have provided an overview of the spatio-temporal stream reasoning architecture designed for ROS, and have presented an in-depth analysis for providing spatio-temporal functionalities to the formula progressor. We have implemented a solution, that extends the original formula progressor with spatial reasoning functionalities, based on the RCC-8 calculus. The result implementation is capable of evaluating spatio-temporal logic formulas using progression over streaming data. Furthermore, ROS im-

plementation of the formula progressor was presented as a Stream Reasoner which is a part of a spatio-temporal stream reasoning architecture in ROS.

Through performance evaluation we have shown that our solution is efficient and scalable. We have evaluated our solution based on different aspects of the system that effect the execution time. The experiments produced the expected results. We have shown that the formula progressor is fast and is capable of handling large number of formulas and formulas with large number of terms. The slower part of the process is the spatial reasoning and the size of the spatial knowledge base in particular. However, this as well was expected, as the results conform with the time complexity of the path-consistency algorithm used.

7.2 Discussion

In chapters 4 and 5 we have presented our solution and implementation and motivated the choices, assumptions and directions selected to overcome some of the problems. However, there are further possibilities to explore in order to further improve the spatio-temporal stream reasoning. The main areas of possible improvement are dealing with indefinite knowledge, in particular finding ways of reducing it, and adapting the spatial reasoner to the stream reasoning and temporal approach and dealing with incremental data.

In our implementation we rely on GQR for the spatial reasoning. However, GQR is a generic CSP solver and is not designed to support incrementally and continuously changing constraints. In the following part we present some suggestions for modifying the reasoning procedure based on three different perspectives relevant to our use-case scenarios and implementation:

- Stream reasoning perspective - Taking into consideration the streaming nature of the incoming data and handling incremental state updates.
- Spatio-temporal reasoning perspective - Using additional rules based on spatio-temporal reasoning theory to filter relations, and modify the constraint network to reduce indefinite knowledge.
- UAV domain perspective - Using certain properties of the domain to add additional constraints, filter relations based on different scenarios, or to influence the consistency check.

In the following subsections we suggest some approaches for modifying the implementation. The approaches are grouped based on the perspective, but note that most of the different approaches suggested are independent from the others and could be possibly combined together.

7.2.1 Stream reasoning perspective

In our system we deal with incrementally incoming and continuously changing data. GQR performs the reasoning on a constraint network which represent a static state and the consistency of the full constraint network is recalculated after every change in the spatial relations. However, in reality most of the relations between regions do not change between states. Therefore we can reuse the previous knowledge, namely the values for the relations between the regions in the previous state, when checking the consistency of the new state.

To achieve this we can keep track of the previous value of the relations between each pair of regions. As new state samples arrive, if a relation has been changed for a pair of regions, then the relations between each of those regions and all other regions should be examined. If the relations have not changed, there is no need to examine further. This should be done within the path consistency algorithm, so when propagating the constraints if the current constraint is the same as in the previous state, it should not be added to the queue for consistency checking any further.

7.2.2 Spatio-temporal reasoning perspective

In the presented implementation the spatial and temporal reasoning are done independently. First, the spatial terms are evaluated and then the formula is progressed further. However, we can take advantage of the existing spatio-temporal reasoning theory, in particular the continuity network of RCC-8 and include it in the spatial reasoning procedure.

The continuity network presents rules about the transitions of the relations between regions. It defines the possible relations a pair of regions might be in the next state based on the relation at the current state.

This rules can be included within the spatial reasoning for either filtering the possible relations, or checking if the received or inferred spatial relations are correct. In order to be possible to check or enforce the continuity network rules, the relations between the regions in the previous state need to be kept. One way to integrate this is to check the new relations and compare with the previous ones to filter relations that are not possible. This would reduce indefinite knowledge in the spatial knowledge base. Since removing some relations have implications in the further propagation of constraints, this should be implemented within the consistency check as it affects propagation of values through the constraint network. Another way to integrate this, on another level though, is to define set of formulas within the progressor that need to be true at every step. The continuity network of RCC-8 can be encoded in ST_0 [Wolter and Zakharyashev, 2000], so this approach can be used to check if these formulas are true for every state during the progression. This way the indefinite knowledge in the spatial knowledge base is not reduced, but received and inferred spatial relations can be checked if they are correct.

There are certain assumptions to be made with this approach. Most importantly, a realistic situation is that a change in a relation may be skipped, due to different possible reasons. This could possibly lead to inconsistent or incorrect state of the constraint network in the first case, or some of the rule formulas failing in the second case.

Regarding dealing with indefinite knowledge in our implementation (section 4.3), there is another possible approach to reduce the number of false positives. The idea is to take into consideration the logical operators between the spatial terms when generating consistent scenarios. Let us consider an example, such that a monitor formula contains $DC(uav, restricted) \wedge DC(uav, road)$. Let us further assume that there are multiple possible relations for both pairs that contain the relation DC for both pairs of regions. Instead of generating two consistent scenarios separately for each pair and then evaluating the conjunction of both terms, a single consistent scenario can be tried for a constraint network where the values for both pairs of regions are set to the ones in the formula. However, when dealing with disjunctions between spatial terms in the formula this is not needed. Nevertheless, this approach does not completely solve the uncertainty issue with the result when a term or formula is evaluated to true, because it is not possible to be certain about the relation due to insufficient knowledge about the spatial environment or due to imprecise input.

7.2.3 UAV domain perspective

There are additional implementation details that might be considered when focusing on a specific domain. Certain constraints exist in some domains that need to be taken into consideration when progressing states or when checking consistent relations. In our case, we focus on the UAV domain and discuss scenarios relevant for the domain.

In the UAV domain, we might want to consider the UAVs as atomic units, meaning a region that represents a UAV is the smallest region possible in our environment. Therefore, one obvious constraint could be that no other regions can be in $NTTP$ or TPP relations with an UAV, and inversely the UAV cannot be in $NTPPI$ and $TPPI$ relations with other regions. Enforcing these constraints could reduce indefinite knowledge. This approach can be generalized for different regions, not just UAVs. One possible way to do this is to define different types of regions, and then have specific constraints for the different types.

In our scenarios the spatial knowledge base represents the state of the spatial environment, in particular spatial regions that represent objects and regions from the outside world. Many of these regions are static, taken from the GIS data for example, and only some regions represent the UAVs, vehicles or other objects that move, or regions that change their size or position. The static objects, do not change the spatial relations between them. If two buildings are disconnected (DC) we can assume that they will

remain like that. Therefore we can define static and dynamic regions in the spatial knowledge base. The dynamic regions are expected to move or change size, therefore are expected to change the spatial relations relative to other dynamic and static regions. Assuming the initial or previous state of the spatial knowledge base is known and available, then the spatial reasoning can be modified to take into consideration only pairs of regions that might have changed relations and ignore pairs of regions that are considered as impossible to change relation. Making a distinction between static and dynamic regions, can limit the path consistency algorithm to consider only dynamic regions, since the relations between the static regions are known and there is no need to recalculate the consistency for the relations between those regions.

This approach can be combined with the approach suggested from the stream reasoning perspective (section 7.2.1)

A test implementation where only the relations between pairs of dynamic regions and pairs consisting of one dynamic and one static regions were checked for consistency, while pairs of static regions were ignored. However, this approach needs to be analyzed further and the implementation needs to be tested, so it is recommended as a future work.

7.3 Future work

There are several directions that can be further explored in order to improve or extend the current solution.

- Implement some of the suggestions presented in the previous section. The presented suggestions focus on improving the performance of the spatial reasoning by either reducing the indefinite knowledge in the spatial database or by adapting the reasoner to streaming data.
- Add support for different spatial calculi. Since GQR is a generic constraint solver and supports different spatial calculi it would be possible to provide additional support for spatial reasoning for other aspects of space, such as orientation. GQR already contains specifications for some calculi, but other calculi can also be specified. Implementation-wise this would require detecting the calculi-specific terms in the formula and evaluate the values of those terms via calls to the external reasoner as done with RCC-8 terms.
- Add support for the ST_1 and ST_2 languages [Wolter and Zakharyashev, 2000]. This would provide capabilities for reasoning not just about changes between regions over time, but changes in the regions themselves over time. However, this would require some sort of progression over temporally quantified region variables.

Bibliography


- N. Asher and L. Vieu. Toward a geometry of common sense: A semantics and a complete axiomatization of mereotopology. In *IJCAI (1)*, pages 846–852, 1995.
- F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *AAAI/IAAI, Vol. 2*, pages 1215–1222, 1996.
- P. Balbiani, J.-F. Condotta, and L. F. del Cerro. A model for reasoning about bidimensional temporal relations. In *KR/98. Proc of the 6th Int. Conf.*, pages 124–130. Morgan Kaufmann, 1998.
- P. V. Beek and D. W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4:1–18, 1996.
- B. Bennett. Spatial reasoning with propositional logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference (KR94)*, pages 51–62. Morgan Kaufmann, 1994.
- B. Bennett. Modal logics for qualitative spatial reasoning. *Logic Journal of IGPL*, 4(1):23–45, 1996.
- B. Bennett and A. G. Cohn. Multi-dimensional multi-modal logics as a framework for spatio-temporal reasoning. In *in Proceedings of the ‘Hot Topics in Spatio-Temporal Reasoning’ workshop, IJCAI-99*, 1999.
- B. Bennett, A. Cohn, F. Wolter, and M. Zakharyashev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, 2002.
- B. L. Clarke. A calculus of individuals based on “connection”. *Notre Dame J. Formal Logic*, (22), 1981.
- A. Cohn, B. Bennett, J. Gooday, and N. Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997a.

- A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundam. Inf.*, 46:1–29, January 2001.
- A. G. Cohn and J. Renz. Qualitative Spatial Representation and Reasoning. *Handbook of Knowledge Representation*, pages 551–596, 2008.
- A. G. Cohn, N. M. Gotts, Z. Cui, D. A. Randell, B. Bennett, and J. M. Gooday. Exploiting temporal continuity in qualitative spatial calculi. *Spatial and Temporal Reasoning in Geographical Information Systems Elsevier*, (May):1–25, 1997b.
- J.-F. Condotta, G. Ligozat, and M. Saade. The qat: A qualitative algebra toolkit. In *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, volume 2, pages 3433–3438, 0-0 2006a.
- J.-F. Condotta, G. Ligozat, and M. Saade. A generic toolkit for n-ary qualitative temporal and spatial calculi. In *Temporal Representation and Reasoning, 2006. TIME 2006. Thirteenth International Symposium on*, pages 78–86, june 2006b.
- K. Conley. ROS Introduction, 2011. URL <http://www.ros.org/wiki/ROS/Introduction>. Last accessed 11th Nov 2011.
- R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. ISBN 978-1-55860-890-0.
- P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19:332–377, 2009.
- Z. Dragisic. Semantic matching for stream reasoning, 2011.
- I. Düntsch, H. Wang, and S. McCloskey. A relation-algebraic approach to the region connection calculus. *Theor. Comput. Sci.*, 255:63–83, March 2001.
- M. J. Egenhofer. Reasoning about binary topological relations. In *Proceedings of the Second International Symposium on Advances in Spatial Databases, SSD '91*, pages 143–160, London, UK, 1991. Springer-Verlag. ISBN 3-540-54414-3.
- A. Frank. Qualitative spatial reasoning about cardinal directions. *Technical papers ACSM-ASPRS annual convention, Baltimore, 1991. Vol. 6: AUTO-CARTO 10*, pages 148–167, 1991.
- C. Freksa. Using orientation information for qualitative spatial reasoning. In *Proceedings of the International Conference GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning on Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 162–178, London, UK, 1992. Springer-Verlag. ISBN 3-540-55966-3.

- Z. Gantner, M. Westphal, and S. Wöflf. Gqr - a fast reasoner for binary qualitative constraint calculi. volume WS-08-11, pages 24–29, 2008.
- A. Gerevini and B. Nebel. Qualitative spatio-temporal reasoning with rcc-8 and allen’s interval calculus: Computational complexity. In *ECAI’02*, pages 312–316, 2002.
- A. Gerevini and J. Renz. Combining topological and size information for spatial reasoning. *Artificial Intelligence*, 137:1–42, May 2002. ISSN 0004-3702.
- R. Goyal and M. Egenhofer. Consistent queries over cardinal directions across different levels of detail. *Database and Expert Systems Applications, International Workshop on*, 0:876, 2000. ISSN 1529-4188.
- R. K. Goyal and M. J. Egenhofer. Similarity of cardinal directions. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases, SSTD ’01*, pages 36–58, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42301-X.
- F. Heintz. *DyKnow: A Stream-Based Knowledge Processing Middleware Framework*. PhD thesis, PhD Dissertation, Linköpings universitet. Linköping Studies in Science and Technology, Dissertation No 1240, 2009.
- D. Hernández. *Qualitative Representation of Spatial Knowledge*, volume 804 of *Lecture Notes in Computer Science*. Springer, 1994. ISBN 3-540-58058-1.
- Z. Ibrahim and A. Tawfik. An abstract theory and ontology of motion based on the regions connection calculus. volume 4612 LNAI, pages 230–242, 2007.
- O. Jihong, F. Qian, and L. Dayou. A model for representing topological relations between simple concave regions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4487 LNCS:160–167, 2007.
- R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2:255–299, October 1990.
- J. Kvarnström and P. Doherty. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30: 119–169, June 2000.
- G. Ligozat. Reasoning about cardinal directions. *Journal of Visual Languages and Computing*, 9(1):23–44, 1998.
- W. Liu, S. Li, and J. Renz. Combining rcc-8 with qualitative direction calculi: Algorithms and complexity. pages 854–859, 2009.

- W. Liu, X. Zhang, S. Li, and M. Ying. Reasoning about cardinal directions between extended objects. *Artificial Intelligence*, 174(12-13):951 – 983, 2010.
- R. Moratz, F. Dylla, and L. Frommberger. A relative orientation algebra with adjustable granularity. In *Proceedings of the Workshop on Agents in Real-Time and Dynamic Environments (IJCAI 05)*, 2005.
- P. Muller. A qualitative theory of motion based on spatio-temporal primitives. *Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference (KR-98)*, pages 131–141, 1998.
- P. Muller. Topological spatio-temporal reasoning and representation. *Computational Intelligence*, 18(3):420–450, 2002.
- B. Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ord-horn class. *Constraints*, 1, 1997.
- D. Papadias and Y. Theodoridis. Spatial relations, minimum bounding rectangles, and spatial data structures. *International Journal of Geographic Information Science*, 11:111–138, 1997.
- M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. *3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176, 1992.
- J. Renz. *Qualitative spatial reasoning with topological information*. Lecture notes in computer science. Springer, 2002. ISBN 9783540433460.
- J. Renz and G. Ligozat. Weak composition for qualitative spatial and temporal reasoning. In P. van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 534–548. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29238-8.
- J. Renz and D. Mitra. Qualitative direction calculi with arbitrary granularity. In *In Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence*, pages 65–74. Springer, 2004.
- J. Renz and B. Nebel. Efficient methods for qualitative spatial reasoning. In *Proceedings of the 13th European Conference on Artificial Intelligence*, pages 562–566. Wiley, 1998.
- J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: a maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1):69–123, 1999.

- J. Renz and B. Nebel. Qualitative spatial reasoning using constraint calculi. In *Handbook of Spatial Logics*, pages 161–215. Springer, 2007. ISBN 978-1-4020-5586-7.
- S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd edition)*. Pearson Education, 2003. ISBN 0137903952.
- C. Schlieder. Representing visible locations for qualitative navigation. In *Qualitative Reasoning and Decision Technologies*, pages 523–532. CIMNE, 1993.
- C. Schlieder. Reasoning about ordering. In A. Frank and W. Kuhn, editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 988 of *Lecture Notes in Computer Science*, pages 341–349. Springer Berlin / Heidelberg, 1995. ISBN 978-3-540-60392-4.
- S. Skiadopoulos and M. Koubarakis. On the consistency of cardinal direction constraints. *Artificial Intelligence*, 163(1):91–135, 2005.
- L. Vieu. Spatial representation and reasoning in artificial intelligence. In *Spatial and Temporal Reasoning*, pages 5–41. Springer Netherlands, 1997. ISBN 978-0-585-28322-7.
- J. O. Wallgrün, L. Frommberger, D. Wolter, F. Dylla, and C. Freksa. Qualitative spatial representation and reasoning in the sparq-toolbox. In *Proceedings of the 2006 international conference on Spatial Cognition V: reasoning, action, interaction*, SC’06, pages 39–58, Berlin, Heidelberg, 2007. Springer-Verlag.
- F. Wolter and M. Zakharyashev. Spatio-temporal representation and reasoning based on rcc-8. In *In Proceedings of the seventh Conference on Principles of Knowledge Representation and Reasoning, KR2000*, pages 3–14. Morgan Kaufmann, 2000.
- F. Wolter and M. Zakharyashev. *Qualitative spatiotemporal representation and reasoning: a computational perspective*, pages 175–215. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. ISBN 1-55860-811-7.

	Avdelning, Institution Division, Department AIICS, Dept. of Computer and Information Science 581 83 Linköping	Datum Date 2012-02-10
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN - ISRN LIU-IDA/LITH-EX-A-12/008-SE Serietitel och serienummer ISSN Title of series, numbering -
URL fr elektronisk version http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-75885		Linköping Studies in Science and Technology Thesis No. LIU-IDA/LITH-EX-A-12/008-SE
Titel Title Extending the Stream Reasoning in DyKnow with Spatial Reasoning in RCC-8 Författare Author Daniel Lazarovski		
Sammanfattning Abstract <p>Autonomous systems require a lot of information about the environment in which they operate in order to perform different high-level tasks. The information is made available through various sources, such as remote and on-board sensors, databases, GIS, the Internet, etc. The sensory input especially is incrementally available to the systems and can be represented as streams. High-level tasks often require some sort of reasoning over the input data, however raw streaming input is often not suitable for the higher level representations needed for reasoning. DyKnow is a stream processing framework that provides functionalities to represent knowledge needed for reasoning from streaming inputs. DyKnow has been used within a platform for task planning and execution monitoring for UAVs. The execution monitoring is performed using formula progression with monitor rules specified as temporal logic formulas. In this thesis we present an analysis for providing spatio-temporal functionalities to the formula progressor and we extend the formula progression with spatial reasoning in RCC-8. The result implementation is capable of evaluating spatio-temporal logic formulas using progression over streaming data. In addition, a ROS implementation of the formula progressor is presented as a part of a spatio-temporal stream reasoning architecture in ROS.</p>		
Nyckelord Keywords Qualitative Spatial Reasoning, Qualitative Spatio-Temporal Reasoning, RCC-8, DyKnow, Stream Reasoning, ROS		