



Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

**Technical
Memo**

TM-92-04

On the Representation of Temporal Knowledge

Jürgen Müller, Jörg Müller,
Markus Pischel, Ralf Scheidhauer

May 1992

**Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH**

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Philips, SEMA Group Systems, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

On the Representation of Temporal Knowledge

Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer

DFKI-TM-92-04

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9104).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1992

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.

On the Representation of Temporal Knowledge

Jörg Müller, Jürgen Müller, Markus Pischel, Ralf Scheidhauer
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-6600 Saarbrücken 11

May 26, 1992

Abstract

The growing interest in an adequate modelling of time in Artificial Intelligence has given rise to the research discipline of *Temporal Reasoning (TR)*. Due to different views, different approaches towards TR such as PL1, modal logics or Allen's interval logic have been investigated. It was realized at an early stage that each of these approaches has some strong points whereas it suffers from certain drawbacks. Thus recently, a number of research activities have emerged aiming at a combination of the classical paradigms for representing time.

In the first part of this paper, we present an overview of the most important approaches to the integration of temporal knowledge into logic programming. In the second part, we present the CHRONOLOG temporal logic programming language which has been developed to cover the quintessence of the approaches presented before. The third part of the paper describes TRAM, which is an extension of CHRONOLOG to a temporal knowledge representation system. Using TRAM it is possible to represent knowledge depending on time and to reason about this knowledge. TRAM has been conceptually based on a combination of modal logics with Allen's interval logic. We present the *Extended Modal Logics (EML)* which establishes the theoretical framework for TRAM. We define an *operational semantics* and a *horizontal compilation scheme* for TRAM.

Contents

I	Temporal Logic	4
1	Introduction	5
1.1	Time - Reaching out for a Mystery	6
1.2	Synopsis of the Paper	7
2	Representing Temporal Knowledge	8
2.1	FOPL and Time	8
2.2	Allen's Interval Logic	8
2.3	The Modal Logics Approach	10
2.4	Evaluation and Conclusion	12
3	Temporal Logic Programming Languages	13
3.1	Abadi and Manna's TEMPLOG	13
3.2	Dov Gabbay	15
3.3	Kowalski's and Sergots's Calculus of Events	18
3.4	Hrycej's Temporal Prolog	19
3.5	Tang's TPL	21
3.6	Comparison and Conclusion	23
II	CHRONOLOG and TRAM	26
4	An Introduction to CHRONOLOG	27
4.1	The Syntax	29
4.2	The Semantics	30
4.3	CHRONOLOG Examples	30
4.3.1	The Factory Example	30
4.3.2	The blocks world	31
4.4	Conclusion	33

5	The Basic Ideas of TRAM	34
6	An Extended Modal Logic (EML)	36
6.1	The Syntax of EML	36
6.2	A Model-Theoretic Semantics of EML	37
6.3	Integrating Interval Logic	39
7	The Knowledge Representation System TRAM	41
7.1	An Operational Semantics of TRAM	42
7.2	A Horizontal Compilation Scheme for TRAM Programs	44
7.3	Recalling the Loading Dock Scenario	46
8	Conclusion and Outlook	50
A	The TRAM Solution for the Loading Dock Example	51
	Bibliography	59

List of Figures

3.1	A sample TPL program	22
4.1	A CHRONOLOG world graph	29
5.1	A Loading Dock Scenario	35
7.1	A TRAM database for the Loading Dock Scenario	47
7.2	A TRAM Starting World	48
7.3	A Query to the TRAM System	49

Part I
Temporal Logic

Chapter 1

Introduction

Time plays an important role in many real-life problems, and reasoning about time often seems indispensable. Thus, researchers in Artificial Intelligence are faced with the need of finding acceptable representations of time and temporal knowledge. However, it is very difficult to express what we actually mean when talking about time, and what the basic characteristics of time are. Trying to formalize temporal aspects and fitting them into a general framework appears to be even harder. Moreover, time can have many different faces and aspects of topology, which may depend on the different points of view: time can be regarded as *continuous* or *discrete*, as *intervals* or *points*, as *linear*, *branching*, or *parallel*. Nevertheless, the problems related to time do not free us from having to cope with temporal aspects in many fields covered by AI, such as expert systems for medical diagnosis, where time is an important factor when it comes to diagnosing and healing a disease. Other fields are planning and scheduling, where actions and goals have to be coordinated while keeping a given set of temporal constraints satisfied. Our group¹ investigates time from the point of view of multi-agent systems, where it plays a crucial role for coordination processes (amongst others). **Temporal Reasoning**(TR) has turned out to be a separate area of research within AI (cf. [KS86, Hry88, AM89, Tan89, Gab87]). TR deals with the representation of and the inference on temporal propositions. A major goal is to explore the basic characteristics of temporal structures and to find adequate general models of time².

Amongst other approaches, the logic-based approach towards handling time has appeared to be useful due to its solid mathematical foundations, its clear and formal syntax and semantics which allow well-founded statements about soundness, completeness, decidability, computability, efficiency, or complexity etc. Our approach is closely oriented to this logical background, although we do not deny that it suffers from a number of shortcomings which are covered aptly by [SM87] using the well-known metaphore of the man searching his keys under a lantern instead of searching them in the dark where he lost them, because search is simpler in the light of the lantern. However, we think that the advantages of temporal logics finally make up for their drawbacks.

¹This work has been done in the AKA-Mod project at DFKI, Saarbrücken.

²Up to now, approaches dealing with time have been highly specific and depending on the respective applications.

This paper is divided into two parts. In the first part we will give an overview of some recent work in the field of temporal reasoning. In the second part, we will describe the systems CHRONOLOG and TRAM which have been developed at our research center. CHRONOLOG [Sch89] is a temporal logic programming language which extends PROLOG. TRAM [Pis91] is a system for representing temporal knowledge, which combines both interval and point aspects of time by integrating the ideas of two of the main paradigms for the representation of time in AI: Allen's interval logic [All84] and the modal logics approach [Pri67, AM89]. The former approach employs intervals and relations between intervals as the basic entities, whereas the latter one uses a set of modal operators which are interpreted by using a possible worlds semantics in order to express temporal knowledge. In the following, we will outline the most crucial aspects of time and their relevance for research in Computer Science.

1.1 Time - Reaching out for a Mystery

The question of the nature of time is a very difficult one, and no generally accepted answer could be given to point out, what the interesting characteristics of time are and how they could be represented. It depends strongly on the domain in which time constructs are to be used. So the best way to see what aspects time can have is to examine in more detail the various conceptions used in different domains. However, there are a few general characteristic arguments and criteria which are heard very often in discussions about temporal aspects. They are summarized in the following:

Point or interval

Some people think of time as having interval character: properties hold during intervals of time, but one does not know how two different intervals are related to each other (i.e. one interval may surround, overlap, be before, or after another one). The aim of reasoning is to get information about the relationships between the intervals.

Another way is to understand time in a modal logic way: the universe is a graph, whose nodes represent different time points. Edges are drawn due to explicitly or implicitly stated rules of the logic. Here, the aim is to find out if a property that holds in a certain node (which represents the current time) will be true e. g. for one or all succeeding or preceding timepoints.

Discrete or continuous

Out of the point/interval discussion comes the way to see time more as a sequence of time instants or as a continuous flow.

Time instants are used e. g. in program verification or in the blocks world, where time can only change significantly between the execution of two directly succeeding program statements or two actions of the roboter arm.

To look upon time as a continuous flow makes more sense in an environment such as natural language processing: here it is possible that each event that was told in a sentence can have many subevents, told in future sentences. It may also be possible that there appears a third intermediate event between two events which are supposed to be directly succeeding.

Branching or Linear

When reasoning about future and past, people regard time as having branching character, and they try to find out what will be or what had been. Others base on one single time line. They are more interested in *when* things happened and not *if* they will/had happen(ed) at all.

1.2 Synopsis of the Paper

Section 2 shortly describes the state of the art in temporal logics. Some significant formalisms for modelling temporal knowledge are presented and their properties are discussed. In chapter 3, we present some of the most interesting approaches towards building temporal logic programming languages. The evaluation of the strong and weak points of existing approaches will be the starting point to the systems CHRONOLOG and TRAM developed at our institute, which we will present in the second part of this paper. In section 4 we provide a brief outline of the CHRONOLOG temporal logic programming language, which basically constitutes an extension of PROLOG by temporal constructs. In chapter 5 we motivate the development of the TRAM system which shall be looked upon as a combination of different conceptual approaches towards the representation of time, basically the combination of modal logics with Allen's interval logic model. In chapter 6, the theoretic framework of TRAM, the extended modal logic EML, is described. Both the syntax and a model-theoretic semantics for EML are given, and we show how interval logic according to Allen can be integrated into a modal logic framework. Chapter 4 describes the basic features of the knowledge representation system TRAM. We give an operational semantics for TRAM³, and we provide a horizontal compilation scheme for TRAM programs into PROLOG programs. Chapter 8 summarizes the most important results and gives a short outlook.

³Our current TRAM version is based on PROLOG. Thus the operational semantics of TRAM should be intuitive for a reader familiar with PROLOG

Chapter 2

Representing Temporal Knowledge

This chapter will present the framework for a unified model-theoretic view of time. For this purpose, three approaches towards representing temporal knowledge are shortly discussed.

2.1 FOPL and Time

The use of first order predicate logic (FOPL) is very popular for many purposes in AI. Its usefulness for knowledge representation already has been known for a long time (cf. [Nil80, Llo84, BB87, RK91]). The main properties of FOPL are its *semi-decidability* and *completeness*. Since powerful theorem provers for FOPL exist [OS89], an integration of temporal reasoning into FOPL appears to be promising. [Pis91] provides a summary of different approaches towards the representation of temporal knowledge in FOPL.

However, most researchers agree that FOPL does not provide optimal support to temporal aspects. As a consequence of this, FOPL has been extended in many ways in order to gain more expressive and more powerful formalisms to represent and to reason about temporal knowledge. One of them is Allen's interval logic, which we will discuss in section 2.2, another one is the modal logics approach (cf. section 2.3).

2.2 Allen's Interval Logic

Allen's approach towards the representation of temporal knowledge is doubtlessly one of the most prominent works in this area of research. The origin of Allen's work is the processing of natural language where time and its representation play an especially important role. In this section we give a short summary of the topics of Allen's interval logic. For a more detailed view we refer to Allen's original articles [All83, All84].

In his model, Allen uses *intervals* as the basic entities. He characterizes time by a set of events (intervals) together with several temporal relations such as *before*, *overlaps*, *meets*, *equal*, *during*, *starts*, *finishes* between these intervals. Allen covers all possible constellations between two intervals by introducing 13 relations which are shown in table 2.1. The resulting

structure is a graph which can be regarded as a constraint net, where the consistency of different temporal interdependencies can be checked using constraint propagation.

Relation	Symbol	Symbol for Inverse	Pictorial Representation
X before Y	<	>	X  Y 
X equal Y	=	=	X  Y 
X meets Y	<i>m</i>	<i>mi</i>	X  Y 
X overlaps Y	<i>o</i>	<i>oi</i>	X  Y 
X during Y	<i>d</i>	<i>di</i>	X  Y 
X starts Y	<i>s</i>	<i>si</i>	X  Y 
X finishes Y	<i>f</i>	<i>fi</i>	X  Y 

Table 2.1: Allen's 13 relationships between intervals

Allen uses three meta-concepts *occur*, *holds*, and *occurring* each of which can be applied to one of the following three classes of knowledge items: *events*, *properties* and *processes*. An algebraic structure is defined on top of the relations using two operators, the *intersection* between sets of possible relations between intervals, and the *composition* of sets of relations. Allen has operationalized his concepts by providing both a local and a global constraint propagation algorithm.

For many domains, Allen's model is very intuitive and adequate. The constraint propagation mechanism provides an elegant way to cope with incomplete knowledge. Knowledge can be extended and modified incrementally by using incremental constraint solvers. Moreover, the formal FOPL-like framework facilitates the proof of diverse properties. But there are some serious drawbacks of the approach:

- Due to the special semantics of negation in the Allen logic [Gal87], a sound representation of continuous changes is not provided.

- The underlying model of time is restricted to a linear time axis. Especially, it is not possible to model something like branching time.
- The granularity of classifying knowledge in events, properties, and processes is very coarse. For some applications e.g. a finer semantics of parallelism than the one provided by the Allen model would be necessary.
- Constraint propagation is an expensive method of computation.

2.3 The Modal Logics Approach

The theory of modal logics has its origin in the necessity of expressing both knowledge which always (*necessarily*) holds and knowledge which sometimes (*possibly*) holds. This is achieved by a *possible worlds semantics* where those worlds are considered possible which can be reached from the current world using an accessibility relation. The possible worlds together with the accessibility relation establish a graph of worlds, a so called *Kripke structure*. Two modal operators \Box and \Diamond are introduced in order to express validity in all worlds or in some worlds, respectively. For a more detailed introduction to modal logics we refer to [Kri71, Ram88, Sho88].

Modal Logics and Time

By slightly varying standard Kripke semantics, modal logics can be used in order to model temporal knowledge. Worlds are considered as time points, and the semantics of the modal operators is enriched by the notions of *future* and *past*:

- Fp : p holds in some¹ future world.
- Gp : p holds in any future world.
- Pp : p holds in some past world.
- Hp : p holds in any past world.

These operators are interpreted in a Kripke-like manner. Prior [Pri67] was the first to apply the principles of modal logics to tense logics. He showed that the logic defined by the above operators can be considered equivalent to an S4 modal logic².

The strong points of the modal logics approach are obvious. Compared to FOPL, temporal knowledge can be formulated in a very elegant and compact manner, it is possible to represent objects which exist only temporarily. Moreover, it offers powerful mechanisms of representation, e.g. axioms like $\Box Q \rightarrow \Box\Box Q$ cannot be axiomatized in FOPL. However, this approach has some shortcomings:

¹i.e. at least in one

²The crucial property of the S4 axiomatization is that the accessibility relation is transitive and reflexive, whereas it is not symmetric. Obviously, since most of us cannot simply travel back to the past, this seems reasonable for the representation of time.

- Modal temporal logics just gives an indirect representation of time. This makes it hard to refer to subjects involving explicit time.
- Reasoning over intervals of time is not supported by the model.
- Due to combinatorial explosion of the graph of worlds, existing systems using modal logics suffer from efficiency problems.
- Some important properties of worlds (e.g. reflexivity of the accessibility relation) can only be formulated as axioms. Thus, properties valid for individual worlds cannot be expressed in an adequate manner.

Moszkowskis ITL

Moszkowskis interval temporal logic (ITL) is given as an example to clarify the concepts of modal logic. His domain is the behaviour of electrical circuits. He calls his worlds *intervals*, where an interval I is represented as a nonempty sequence of immediately succeeding time points, written as $I = \langle t_0, t_1, \dots, t_n \rangle$. Two intervals are connected in the time graph, if one is a *terminal* subinterval of the other; i.e. $\langle s, t, u \rangle$ has successors $\langle s, t, u \rangle, \langle t, u \rangle$ and $\langle u \rangle$. Formulas in ITL are FOPL formulas with four additional operators: \Box, \Diamond, \bigcirc and $;$ (the *chop* operator). In the following $I \models w$ means that formula w holds during interval I . The new operators are defined as follows:

- $\langle s_0, \dots, s_n \rangle \models \Box w$ iff $\langle s_i, \dots, s_n \rangle \models w$
for all i with $0 \leq i \leq n$
- $\langle s_0, \dots, s_n \rangle \models \Diamond w$ iff $\langle s_i, \dots, s_n \rangle \models w$
for at least one i with $0 \leq i \leq n$
- $\langle s_0, \dots, s_n \rangle \models \bigcirc w$ iff $\langle s_1, \dots, s_n \rangle \models w$
- $\langle s_0, \dots, s_n \rangle \models w; w'$ iff
 $\langle s_0, \dots, s_i \rangle \models w$ and $\langle s_i, \dots, s_n \rangle \models w'$
for at least one i with $0 \leq i \leq n$

Moszkowski uses these constructs to describe the behaviour of digital circuits: for example if one wants to state that two bit signals X and Y are equal over time, one can do this by:

$$X \approx Y \equiv_{def} \Box(X = Y)$$

The \bigcirc operator can be used to represent unit delay for example: if one bit signal X is continuously assigned to another bit signal Y over time but with unit delay one would define it as:

$$X \text{ del } Y \equiv_{def} \Box((X = 0) \equiv (Y = 0))$$

2.4 Evaluation and Conclusion

In this section we presented some important approaches towards representing temporal knowledge: FOPL, Allen's interval logic, and modal tense logic. It has been shown that each of these approaches has some strong points whereas it suffers from some shortcomings. For many applications it can be extremely useful to combine the features of different approaches in order to gain a more powerful and more expressive formalism for representing time. In our work, we decided to integrate elements of Allen's work into a modal logic. Especially, it shall be possible to reason about intervals of worlds and their interdependence. Thus, we enhance the modal time model, which *per se* reveals a time point character of time, by intervals of time. However, we preserve the basic desirable properties of modal logics.

Thus, in the following, after giving a review on some current approaches towards temporal logic programming, we will introduce the CHRONOLOG system which embodies the modal logics approach towards tense logics (see chapter 4. In chapter 6 we will present an extended modal logics (EML) which embodies both modal logics and an interval concept. The knowledge representation system TRAM described in chapter 7 basically constitutes the operationalization of EML.

Chapter 3

Temporal Logic Programming Languages

In this section we present some current proposals for temporal logic programming languages. All issues discussed here are all closely related to PROLOG, because it seems the most promising way for different reasons to take PROLOG and extend it by various temporal constructs: First of all PROLOG is already *programming in logic* and therefore “only” the temporal aspects must be added. Furthermore there has already been done world wide much work both on the theoretical and on the practical side of PROLOG. This results in powerful PROLOG systems (available for all computer systems) that are comparable to i.e. LISP environments both in program development tools (editors, debuggers, etc.) and runtime performance and storage consumption. All this can be used to develop temporal programming tools, that are not only of high academical interest, but also of great practical usefulness.

3.1 Abadi and Manna’s TEMPLOG

Abadi and Manna present a programming language called TEMPLOG, which strongly bases on PROLOG and adds 3 modal logic operators: \bigcirc (“next”), \square (“always”) and \diamond (“eventually”); their meaning is explained as follows:

- $\bigcirc P$ means “ P is true at the next time point”
- $\square P$ means “ P is always true (from now on)”
- $\diamond P$ is defined as: $\diamond P \equiv \neg \square \neg P$

In their view time is *linear, discrete* and extends *infinitely* towards future.

A TEMPLOG program is a collection of temporal (Horn) clauses, where a clause may have one of the following forms:

$$H \leftarrow B$$

$$\begin{aligned}\Box H &\leftarrow B \\ \Box(H &\leftarrow B)\end{aligned}$$

For the sake of readability, the last clause, which is also called *permanent* clause, is written as:

$$H \Leftarrow B;$$

the first two clauses are called *initial*.

The heads H of the clauses are *next-atomic* formulas of the form

$$\bigcirc^n P$$

where $\bigcirc^n P$ means \bigcirc applied n times to P and P is a conventional PROLOG atom of the form $p(t_1, \dots, t_n)$.

The body B is a formula that consists of next-atomic formulas, which are connected by \diamond and \wedge (conjunction, also written as $,$). Thus,

$$\begin{aligned}p(a) &\leftarrow \bigcirc q(f(x)) \\ \Box \bigcirc \bigcirc p(b) &\leftarrow \diamond(\bigcirc(q(y) \wedge \diamond \bigcirc r(c)))\end{aligned}$$

are well-formed clauses. The use of \Box is prohibited in bodies and the use of \diamond is prohibited in clause heads in order to reduce computational complexity. But this is not a real restriction, because clauses like $\diamond P \leftarrow Q$ (say “ P is eventually true if Q is true”) or $P \leftarrow \Box Q$ (“ P is true if Q is always true”) are of no practical use.

Because of the additional modal operators there is a modified SLD-Resolution strategy called *temporal SLD-Resolution*: It also starts with a list of goals (a goal has the same form as a body), replaces a goal by the body of an applicable clause and repeats this step until the goal list is empty or until backtracking must occur. But the specific resolution step is different from standard SLD-Resolution. To clarify this, we firstly restrict the possible goals and clauses, so that we have goals of the form

$$\bigcirc^{i_1} G_1, \dots, \bigcirc^{i_n} G_n$$

We can try to apply initial or permanent clauses of the form

$$\begin{aligned}\bigcirc^j H &\leftarrow \bigcirc^{j_1} B_1, \dots, \bigcirc^{j_k} B_k(*) \\ \bigcirc^j H &\Leftarrow \bigcirc^{j_1} B_1, \dots, \bigcirc^{j_k} B_k(**)\end{aligned}$$

where the H, B_i and G_i are atoms. (How to apply clauses of the form $\Box H \leftarrow B$ will be explained later).

If we want to apply one of these two clauses in both cases H and one of the G_i (e. g. G_1) must be unifiable (with most general unifier say θ). Additionally, if we want to apply (*), then it must hold that

$$i_1 = j$$

and the resolving goal list is:

$$\bigcirc^{j_1} B_1 \theta, \dots, \bigcirc^{j_k} B_k \theta, \bigcirc^{i_2} G_2 \theta, \dots, \bigcirc^{i_n} G_n \theta$$

If we want to apply (**), we have to note that

$$\bigcirc^j H \Leftarrow \bigcirc^{j_1} B_1, \dots, \bigcirc^{j_k} B_k$$

implies

$$\bigcirc^{i_1} H \Leftarrow \bigcirc^{j_1+(i_1-j)} B_1, \dots, \bigcirc^{j_n+(i_n-j)} B_n$$

but only if $i_1 \geq j$.

Thus, we can perform the same resolution step as in the first case leading to the new goals

$$\bigcirc^{j_1+(i_1-j)} B_1 \theta, \dots, \bigcirc^{j_k+(i_k-j)} B_k \theta, \bigcirc^{i_2} G_2 \theta, \dots, \bigcirc^{i_n} G_n \theta$$

If we want to lift this restricted version to the full version, we have to look next at how to apply clauses that have \square in their head:

$$\square H \Leftarrow B$$

which is equivalent to the following two clauses

$$\begin{aligned} H &\Leftarrow r(x_1, \dots, x_n) \\ r(x_1, \dots, x_n) &\Leftarrow B, \end{aligned}$$

where x_1, \dots, x_n are the free variables occurring in B . So we can reduce this case to the previous one.

Now, only the \diamond operator occurring in goals (and bodies, which is the same problem) is still missing. The solution to this employs similar tricks as in case of the \square operator, but needs more time to explain and is therefore omitted.

Note that in all cases applying a clause to a goal leads to a *unique* succeeding goal list and so the branching factor in the search space is determined by the number of clauses. In this way *no additional branching* is obtained by the resolution procedure. An additional advantage is that it can be easily implemented in PROLOG. But it is *difficult to embed* it into existing PROLOG systems, because one needs to modify the available PROLOG interpreters/compiler, which is practically difficult and mostly impossible. Another point of criticism is that there is no concept for branching future *and* past.

3.2 Dov Gabbay

Gabbay extends PROLOG in a way which is closely related to the way Abadi and Manna do; but Gabbay has not only modal operators for the future, but also for the past. He adds two modal logic operators: F (future) and P (past) which also allow to express \square and \diamond :

- Fq means “ q will be true” (including now)

- Pq means “ q has been true”
- $\diamond q = q \vee Fq \vee Pq$
- $\square q = \neg \diamond \neg q$

A program is a collection of clauses, where F and P may occur either in the body or in the head. Since $FFq = Fq$ and $PPq = Pq$, we assume that there are no multiply applied F 's and P 's. Thus, clauses are of the form:

$$H \leftarrow B$$

$$\square(H \leftarrow B)$$

where the first one is called *ordinary* clause (because it is only valid now) and the second one is called *always* clause (because it may be applied in the past, now and in the future). The head H may either be an atomic formula or it may have the form FA or PA , where A may not only be an atom, but it can also be a whole ordinary clause. A body is of the form $A, B \wedge B', FB$ or PB , where A is an atomic formula and B, B' are bodies.

Program execution is driven – as in classical PROLOG – by a list of goals, where a goal has the same form as a body. The first goal is chosen and replaced by the body of an applicable clause. But what “applicable” means is different here, because a head of a clause may contain a whole clause itself. To explain this, we firstly introduce a notation

$$\mathbf{P} ? G = 1(\theta)$$

where \mathbf{P} is a set of clauses (or say a program). G is a goal and θ a substitution. The whole expression means: “The goal G is derivable from program \mathbf{P} under the substitution θ ”.

If we have a goal list $\mathbf{P} ? G_1, \dots, \mathbf{P} ? G_n$ and we want to replace the first one by applying a clause of \mathbf{P} , what the successive goals look like depends on the particular form of G_1 :

1. If G_1 is an atom, then we look for an (ordinary or always) clause, whose head H is unifiable with G_1 via θ . We derive the new subgoal $B\theta$ (where B is the body of the clause) as in ordinary PROLOG.
2. If $G_1 = G'_1 \wedge G''_1$ then use

$$\mathbf{P} ? G_1 = \mathbf{P} ? G'_1 \wedge G''_1 = 1(\theta) \quad \text{iff} \quad \mathbf{P} ? G'_1 = 1(\theta) \quad \text{and} \quad \mathbf{P} ? G''_1 = 1(\theta)$$

Note that in both subgoals there must be the same substitution θ .

3. $G_1 = PA$ is symmetrically to the next case.
4. $G_1 = FA$ is the most complex case, which is rather technical to explain in detail, so we will outline only the main idea.

FA says that if we want to know if A will be true in the *future* (using clauses, that are valid *now*). We try to show this by applying an ordinary or an always clause. The simplest case is, when we have a clause of the form

$$FH \leftarrow B$$

where A and H are unifiable, thus we only get the new subgoal $\mathbf{P}?B$. To apply this clause, we can try to show that H implies (and not only unifies) A in some future state (written as $\{H\}?A$), if we afterwards succeed in proving B , we can argue: B implies that H will be true in some future time and at that time A becomes true too (from H); so A becomes true in the future, say FA is true now.

An analogous way to show FA is to show that A will be true not at the time when H comes true, but after this it will be true (written as $\{H\}?FA$).

These were the possibilities in applying ordinary clauses. Using always clauses one can try the above ways and additional ones. This stems from the fact that an always clause is not only valid now but also in future states. Trying to apply $\Box(H \leftarrow B)$ or $\Box(FH \leftarrow B)$ one need not only to show B , but can also try to show that B will be true (FB), resulting in FH or FFH respectively.

Using all this one can get different argumentation chains to show FA :

$$\begin{aligned}
 B\theta &\rightarrow FH\theta \rightarrow FA\theta \\
 B &\Rightarrow FH = FA \\
 B &\Rightarrow FH \Rightarrow FFA = FA \\
 FB &\Rightarrow FH \Rightarrow FA \\
 FB &\Rightarrow FH \Rightarrow FFA \\
 FB &\Rightarrow FFH \Rightarrow FFA \\
 FB &\Rightarrow FFH \Rightarrow FFFA
 \end{aligned}$$

Except the first one our successive goal list looks as follows:

$$\{H\}?X, \mathbf{P}?B, \mathbf{P}?G_2, \dots, \mathbf{P}?G_n$$

where X equals A or FA . Note however that, if we want to show that in the future H implies X , we may not use all clauses of our program \mathbf{P} , because not all clauses in \mathbf{P} are valid forever. Rathermore, we may use more than simply $\{H\}$. We may use all always clauses of \mathbf{P} ; ordinary clauses must be prefixed with the past operator P . So in the above goal list we can replace $\{H\}?X$ by:

$$\{H\} \cup \{\text{all always clauses of } \mathbf{P}\} \cup \{Pc | c \text{ an ordinary clause of } \mathbf{P}\}?X$$

Gabbay's horn logic allows to reason about the future *and* the past and allows more complex clause heads. But this advantage is gained by a great loss of computational efficiency: in every resolution step and for every applicable clause there are up to 7 paths to test. Especially always clauses lead to the highest branching factor. Additionally the criticisms presented in the previous section hold for Gabbay's logic too: there seems to be no easy way to embed his concepts into PROLOG. In addition it seems to be difficult to efficiently implement the "switching database" mechanism (from \mathbf{P} to \mathbf{P}' in two successive goals $\mathbf{P}?A$ and $\mathbf{P}'?B$).

3.3 Kowalski's and Sergots's Calculus of Events

Kowalski and Sergot present an interval based calculus, that is combined by three building blocks: *events*, *properties* and *time intervals*. As the name "event calculus" already states, it concentrates on events: an event is an action that changes the state of the world. For the sake of simplicity we assume that events can be totally ordered on a single linear (not branching) time line, because an event is assumed to take no time (or better say after the start of an event, there may be no other events to start before the first one is finished). In this way two events e and e' are either equal or one happens before the other, written as $e < e'$. Events are the central constructs because if an event has happened it initiates some *properties* to be true later on. Furthermore, it states that some other properties are no longer true. An *interval* of time is the interval that lies between two directly succeeding events. Two functions *before* and *after* are defined that map events to time intervals; intervals are always referenced by one of these two functions. It is important to note, that i.e. *after*(e) does not mean the whole interval from e to infinity, but only the interval from e to the next following interval. Thus:

$$\begin{aligned} & \text{before}(e) = \text{after}(e') \text{ iff} \\ & e < e' \text{ and there exists no } e'' \text{ such that } e < e'' < e' \end{aligned}$$

Kowalski and Sergot give an axiom system which is formulated in horn clauses, and which thus seems easy to be implemented in PROLOG. The most interesting axioms will be presented in the following:

First there is a metapredicate **Holds**, that represents the database:

Holds(T, P) is true iff property P is true during time interval T . Two axioms state that a property P must have been true before an event E if E terminates P , and that P is true after E if E initiates P :

$$\begin{aligned} (A1) \quad & \mathbf{Holds}(\text{before}(E), P) \leftarrow \mathbf{Terminates}(E, P) \\ (A2) \quad & \mathbf{Holds}(\text{after}(E), P) \leftarrow \mathbf{Initiates}(E, P) \end{aligned}$$

The two predicates **Initiates** and **Terminates** constitute the interface to the user: it is supposed that they are implemented by the user to represent the user's database. E. g. in the blocks world domain a roboter arm can stack block X onto block Y if the arm is holding X and no other block is on Y . As a result, the arm is no longer holding X . This can be expressed as follows:

$$\begin{aligned} \mathbf{Terminates}(E, \text{holding}(X)) & \leftarrow \mathbf{NOT} \text{ on}(Z, Y), \text{holding}(X). \\ \mathbf{Initiates}(E, \text{on}(X)) & \leftarrow \mathbf{NOT} \text{ on}(Z, Y), \text{holding}(X). \end{aligned}$$

Another two axioms for **Start**(T, E) state that event E is the start point for interval T :

$$\begin{aligned} (A3) \quad & \mathbf{Start}(\text{after}(E), E) \leftarrow \\ (A4) \quad & \mathbf{Start}(\text{before}(E), E') \leftarrow \mathbf{Is-same}(\text{after}(E'), \text{before}(E)). \end{aligned}$$

where

$$(A5) \quad \mathbf{Is-same}(\text{after}(E), \text{before}(E')) \leftarrow \mathbf{NOT} E < E'' < E'$$

There are two similar axioms for **End**(T, E) (event E is the end point of interval T):

$$(A6) \quad \mathbf{End}(\mathit{before}(E), E) \leftarrow$$

$$(A7) \quad \mathbf{End}(\mathit{after}(E), E') \leftarrow \mathbf{Is-same}(\mathit{after}(E), \mathit{before}(E')).$$

The above axioms only present a restricted subset of the original ones. However, they should be expressive enough to provide the main ideas of the event calculus. In the model, there is no need for a single time line: events may only be partially ordered, thus it is possible that two events cannot always be compared. The full version provides the possibility to state that an event causes different properties to come true which hold for different times; so the world is seen as a whole graph whose nodes are events. An edge is labelled by the properties that hold during the interval established by the two time points of the events they connect. In contrast to the previously discussed approaches this one has the main advantage that it can be directly integrated into PROLOG: the above axioms can be easily formulated in PROLOG; unfortunately the integration is not a total one: properties and events are not stored like other PROLOG predicates, but in the special database predicate **Holds**.

3.4 Hrycej's Temporal Prolog

Hrycej presents a temporal-logic extension of PROLOG that bases completely on Allen's temporal constraint model of time, using time intervals and the 13 relations between intervals as central concepts. The main advantages of his implementation are *efficiency* and *integrability* into available PROLOG systems. This is gained by restricting Allen's original axioms to six axioms for the predicate *Holds*; where *Holds*(P, T) means that P is true in time interval T and *Holds*(P) that P holds without temporal restriction. The six axioms are:

axiom 1: $\mathit{Holds}(P, S) \ \& \ \mathit{subinterval}(T, S) \Rightarrow \mathit{Holds}(P, T)$

If P holds in interval S , it also holds in any subinterval T of S

axiom 2: $\mathit{Holds}(P) \Rightarrow (\forall T)\mathit{Holds}(P, T)$

If P holds without temporal limitation, it also holds in any time interval.

axiom 3: $\mathit{Holds}(P, T) \ \& \ \mathit{Holds}(Q, T) \Rightarrow \mathit{Holds}(P \ \& \ Q, T)$

If both P and Q hold in T , their conjunction also holds in T .

axiom 4: $\mathit{Holds}(P, T) \ \vee \ \mathit{Holds}(Q, T) \Rightarrow \mathit{Holds}(P \ \vee \ Q, T)$

If at least one of A and B holds in T , their disjunction also holds in T .

axiom 5: $\mathit{Holds}(P, S) \ \& \ \mathit{Holds}(\neg P, T) \Rightarrow \mathit{disjoint}(S, T)$

If P holds in S and (*not* P) holds in T , then S and T are disjoint intervals.

axiom 6: $\mathit{Holds}(P, U) \ \& \ \mathit{Holds}(Q, V) \ \& \ \mathit{union}(U, V, T) \Rightarrow$
 $\mathit{Holds}(P \ \vee \ Q, T)$

If P holds in U and Q holds in V , then their disjunction holds in the union of U and V .

The last axiom leads to difficulties in implementation, resulting in two different approaches: the *constraining* and the *non-constraining* approach. This will be explained in more detail later on.

Hrycej's implementation is fully embedded into PROLOG: he adds four (meta-)predicates `constrain_rel`, `in`, `dur` and `mkdur`, which the user can utilize – besides all other user-defined and PROLOG-builtin predicates.

`constrain_rel(I1,I2,S)` can be invoked to declare two intervals `I1` and `I2` (if they are not yet known). `S` is a list containing some of Allen's 13 relations. The relations between `I1` and `I2` are constrained to `S`. Constraint propagation is employed, using `S` to constrain other intervals to each other. For example if one wants to tell Hrycej's system that the interval `morning` has two subintervals `8to10` and `10to12`, one can do this by:

```
:- constrain_rel(morning, 8to10), [di]).
:- constrain_rel(morning, 10to12), [di]).
:- constrain_rel(8to10, 10to12), [m]).
```

Remember that $(I1 \text{ di } I2)$ means `I1` contains `I2`, and $(I1 \text{ m } I2)$ means `I1` meets `I2`.

`in` is a predicate that is supposed to be provided by the user, where `in(P,T)` declares that `P` only holds during interval `T`. Here `P` can be a fact, but might also be a rule.

Example:

```
is_to_speak(X) :- at_home(X).
in(is_to_speak(X):- at_work(X), working_time).
in(at_home(tom), morning).
```

This declares that everyone is either always to speak, if he is at home or he is to speak at work but only during working time. The last clause tells us, when `tom` is at home. Here `working_time` and `morning` are time intervals, that must be declared by `constrain_rel`. Depending on the context one may want to be less restrictive, only saying, that they should not be disjoint:

```
:- constrain_rel(morning, working_time,
                [o,oi,d,di,s,si,f,fi,=]).
```

`dur` and `mkdur` implement the above axioms. There are two predicates because of the difficulties already mentioned which arise because of axiom 6. The other axioms can be implemented more easily; for example axiom 1 looks in PROLOG like:

```
dur(P,T) :- in(P,S), subinterval(S,T).
```

Axiom 2 is written as:

```
dur(P,T) :- call(P).
```

Here, we present simplified versions, because the original clauses are rather technical and of less interest than the axiom for disjunction: suppose we want to know whether $P \vee Q$ holds during interval I . To solve this one can search for an interval I_P in which P is true and another interval I_Q in which Q is true. Then one must look whether I_P and I_Q overlap, meet or contain each other, and test whether I is a subinterval of the union of I_P and I_Q . This is exactly the way `dur` implements axiom 6. But this way will rarely lead to a solution, because it needs much information about the relationship between known intervals.

`mkdur` goes another way by changing the constraint net. It also looks for I_P and I_Q above. But then it does not test, if I_P and I_Q *do* overlap. Rathermore, it tests whether they *can* overlap: if there is no information in the net that I_P and I_Q cannot overlap, then the constraint net is modified so that I_P and I_Q are supposed to overlap. Computation proceeds until the final solution is found or until backtracking occurs. In the case of backtracking, the net is restored to its previous state and the next choice point is tried.

Although the constraining approach is more powerful than the non-constraining one it also comes with two main disadvantages: Firstly the cut operator (!) can no longer be used in all places: if the cut is executed after changing the net and if backtracking occurs after processing the cut, the net cannot be restored to its previous state, because the predicate that originally changed the net is not backtracked. Secondly, in the net-changing step one must exactly try four different relations between I_P and I_Q to insure that the constructed interval is the maximal one:

1. I_P overlaps I_Q
2. I_Q overlaps I_P
3. I_P contains I_Q
4. I_Q contains I_P

This leads to a rapid combinatorial explosion.

3.5 Tang's TPL

Tang's TPL is a modal logic extension to PROLOG that differs from PROLOG in two major ways: First of all in TPL there is not only one single database, but there is a whole set of databases DBS , where every database represents a single time point. Additionally, there must exist a predefined relation $R : DBS \times DBS$, to connect the different databases. The second extension of TPL to PROLOG consists of several predefined modal logic meta-predicates which can be used as goals in bodies of user defined clauses to switch between the different databases. Here we only present two of them, **EX** and **AX**:

- **AX**(P) succeeds if P succeeds in *all* worlds (databases) that follow the current one.
- **EX**(P) succeeds if P succeeds in *any* world that follows the current one.

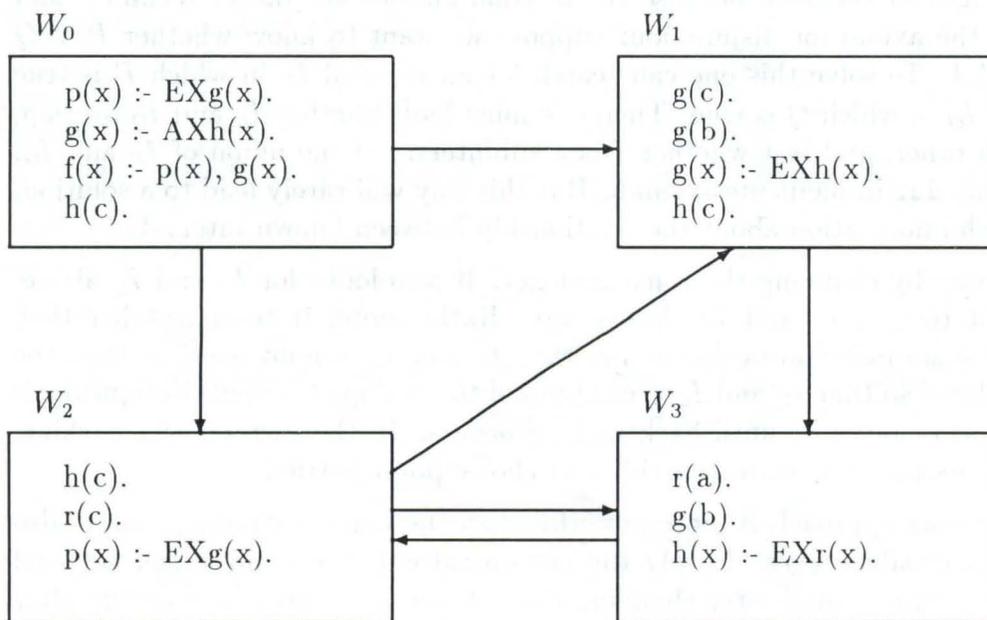


Figure 3.1: A sample TPL program

A sample TPL program is given in figure 3.1

A TPL program is directly represented in PROLOG as follows: every TPL atom gets an additional argument representing the TPL database it belongs to. The relation R is directly represented by a PROLOG predicate called `world`. The TPL program of figure 3.1 will be written as:

```

world(w0,w1).
world(w0,w2).
world(w2,w1).
world(w2,w3).
world(w1,w3).
world(w3,w2).

p(X,w0) :- EX(g(X),w0).
g(X,w0) :- AX(h(X),w0).
f(X,w0) :- p(X,w0), g(X,w0).

g(c,w1).
g(b,w1).
g(X,w2) :- EX(h(X),w1).

h(c,w2).
r(c,w2).
p(X,w2) :- EX(g(X),w2)).

r(a,w3).
g(b,w3).

```

```
h(X,w3) :- EX(r(X),w3).
```

Now the implementation of EX and AX can be easily derived from the above. For example EX:

```
EX(P,W) :- world(W,Wnext),  
            P =.. L,  
            append(L, Wnext, Lnew),  
            Pnew =.. Lnew,  
            call(Pnew).
```

Here the goals 2 to 4 in the body only serve to add *Wnext* as an additional argument to P. If we have the goal

```
?- f(X,w0).
```

then we can infer $X=c$ by the following resolution steps:

```
f(X,w0)  
p(X,w0), g(X,w0)  
EX(g(X),w0), g(X,w0)  
g(X,w1), g(X,w0)  
g(c,w1), g(c,w0)  
g(c,w0)  
h(c,w1), h(c,w2)  
h(c,w2)
```

In the original paper, Tang defines also some additional modal logic operators, whose implementation is a bit more tricky than EX or AX. Additionally he gives a detailed semantical description of TPL using Buechi automata.

Tang's approach is (like Hrycej's) easy to embed in any existing PROLOG system. The search space strongly depends on the size of the world graph. Unfortunately, Tang does not provide concrete examples on how he uses TPL in a practical domain; he only states that he uses TPL to verify automatically some concurrent programs which process infinite states.

3.6 Comparison and Conclusion

Up to now in this chapter we presented five approaches, that all extend PROLOG in their own direction. Now the question arises: What is the best extension? Every issue has its own advantages and disadvantages. The main criterion how to divide them is the **time model** they use. Both Hrycej and Kowalski & Sergot use a time model that concentrates on *intervals* of time, whereas the three others see time as having punctual or *pointwise* character.

Although Kowalsky and Hrycej both use intervals, they differ a little in their **central concepts**: Hrycej concentrates on the possible relations between the known intervals and uses

mostly these relations for reasoning. On the other hand in the center of Kowalsky's & Sergot's approach there are events and properties. Events start and end the validity of certain properties. One event can also start or end different properties, which leads to a **branching model of time** with both branching future and past, in contrast to Hrycej whose model only provides a single time line. The linear time model is also common to the modal logic approach of Abadi & Manna: in TEMPLOG one can only reason about future states and not about future *and* past as in Gabbay's issue. Additionally Gabbay does not necessarily need a linear model of time.

Tang allows a more complex view of the world: here one can directly express what is true in different worlds and how those worlds are connected to each other. This leads to a model of branching time. In the previous two approaches one can only state what is true now or what is always true; other worlds can be referenced by the modal logic operators.

In the **implementations** of the various concepts two groups can be distinguished: Kowalsky & Sergot, Hrycej and Tang take PROLOG and add their own predicates, that can be used besides other PROLOG predicates. Thus they combine the practical advantages of existing PROLOG systems with their own time concepts. The other two approaches define their own languages that syntactically look more or less like PROLOG, but have a different underlying resolution step, such that existing interpreters must be modified or completely reimplemented. Especially Gabbay gives an operational semantics, where it is left unclear how to implement a simple interpreter based on a proof search strategy.

The different modifications to PROLOG result in different **computational complexities**. Gabbay pays for his ability to reason about future and past with a high branching factor during each resolution step. In contrast, in Abadi & Manna the branching factor is determined by the number of applicable clauses as in ordinary PROLOG. Hrycej has performance problems both with constraint propagation and (in the constraining approach) with the different possibilities to constrain two intervals.

But now back to our question: What is the best approach? None of them seems to figure out, what time *really* is, and it is hard to find strong reasons to prefer one over the others in general. It strongly depends on the particular domain to be chosen. For natural language processing e. g. Allen's / Hrycej's approach seems to be of great interest; for solving problems using historical databases, Abadi & Manna's TEMPLOG seems to be the better one.

In summary each particular approach has its own advantages that make it seem superior to the others, but also has disadvantages w. r. t. its competitors. No general solution can be given that combines all advantages and strips off the disadvantages, because time can have so many facets, and it is likely that a general concept would be computationally intractable. Thus, the best way is to study the different issues in detail and afterwards to choose the one that seems to fit best for the particular problem. If none of the above models seems adequate, one needs to develop one's own new calculus. This observation leads to the second part of the paper where we present the CHRONOLOG temporal logic programming language and the TRAM knowledge representation system as integrating approaches towards the handling of temporal knowledge.

Aspect	Abadi & Manna	Gabbay	Kowalsky & Sergot	Hrycej	Tang
time model	point		interval		point
central concepts	practicable modal Horn logic	modal logic with future <i>and</i> past	events and properties	interval relations	Kripke structures
time line	linear	branching future and past	branching future and past	linear	branching time graph
extensions to PROLOG	\bigcirc , \square and \diamond operators initial and permanent clauses	<i>F</i> and <i>P</i> operators ordinary and always clauses	predicates for event handling	EX, AX and other modal operators	predicates: <code>constrain_rel</code> , <code>in</code> , <code>dur</code> and <code>mkdur</code>
Integrability into PROLOG	interpreter modification		fully integrable	fully integrated	
computational complexity	number of clauses	different paths for every clause to test		constraint propagation, constraining intervals	size of the world graph
applications	historical databases		plan generation	nat. language processing	verifying concurrent programs

Table 3.1: Comparison of the different approaches

Part II

Chronolog and Tram

Chapter 4

An Introduction to Chronolog

The temporal logic programming language CHRONOLOG has been developed by Ralf Scheidhauer, and has been described in an unpublished internal report (cf. [Sch89]). In this section we give a short description of the main ideas behind the CHRONOLOG system.

CHRONOLOG is closely related to Abadi & Manna's approach, but extends TEMPLOG in various directions. CHRONOLOG also includes the possibility of defining multiple databases like Tang does and it has the ability to reason about future *and* past as in Gabbay's approach.

TEMPLOG has a single built-in modal operator \bigcirc (NEXT) that generates new unique worlds from an existing initial world. So every world has its own uniquely determined successor. But this might not be enough, because real world tells us that at a certain timepoint many different events can happen leading to different directly succeeding future worlds. In the blocks world e. g. a roboter arm can stack or unstack certain blocks, and different successor states will arise according to the action previously performed. Therefore, CHRONOLOG allows *user definable* modal operators each of which generates a unique new succeeding world from the current one. These operators are not restricted to mere constant symbols, but they may also be compound terms. Thus, it is possible to define a modal operator *stack* with two variable arguments X and Y . In our example it is also necessary to state that *stack* is not applicable in any state, but only when both blocks X and Y are free. This can be done by adding corresponding goals to every clause in the database that deals with stacking. But here CHRONOLOG has an additional feature, that allows *conditional* modal operators. The following clause declares a 2-ary modal operator *stack*, that can only be applied to blocks X and Y , if both are free:

$$stack(X, Y)_W := clear(X)_W, clear(Y)_W$$

The index variable W tells CHRONOLOG, that operator *stack* can be applied to every world. Instead of W we could also write a modality (= a nonempty sequence of terms; read from right to left): thus we could use an initial world w_0 and the modality

$$stack(a, b) unstack(c, d) w_0$$

to denote the world, that results from w_0 by first unstacking block c from d and afterwards stacking a on b .

Goals in CHRONOLOG must always be indexed by a modality:

$$on(a, b)_{W_{w_0}} \leftarrow$$

asks for an operation, that can be applied to world w_0 and results in a world in which a is on b

The user can create a complex world graph using its own modal operators. CHRONOLOG provides additional *meta predicates* that allow to reason about statements that are true in several or all worlds in the *future* and the *past*:

- $(\Box_f P)_W$ means: P will be true in all future worlds of W (including W)
- $(\Diamond_f P)_W$ means: P will be true in some future worlds
- $(\Delta_f P)_W$ means: P will be true in some direct future worlds
- $(\nabla_f P)_W$ means: P will be true in all direct future worlds

$\Box_p, \Diamond_p, \Delta_p, \nabla_p$ are the corresponding equivalents for the past. While those modal operators are built-in, e. g. a user defined modal operator *unstack* can be introduced using one of those built-in operators. If we want to specify that *unstack* should be applied at most once to the same blocks in a sequence of *stack* and *unstack* operations, this can be expressed by defining a modal operator *unstack* with modality variable W :

$$unstack(X, Y)_W := clear(X)_W, clear(Y)_W, (\Box_p on(X, Y))_W$$

In this way CHRONOLOG allows to define multiple databases like Tang does. Every clause gets an index that marks the world in which the clause is to be true.

$$\begin{array}{ll} on(a, b)_{w_0} \leftarrow & block(a)_W \leftarrow \\ on(b, c)_{w_0} \leftarrow & block(b)_W \leftarrow \\ & block(c)_W \leftarrow \end{array}$$

represents that block a is on block b , block b is on c in an initial world w_0 , and that a block is a block in every world.

A special predicate *edge* is a built-in of CHRONOLOG: it is used during reasoning to find out which worlds are connected to each other in the world graph. But the user can also add his own clauses for *edge* to define which worlds should be connected additionally. Suppose you have worlds w_1, w_2, \dots, w_{11} . The following clauses connect them in various ways:

$$\begin{array}{ll} edge(w_2, w_6) \leftarrow & edge(w_9, w_{10}) \leftarrow \\ edge(w_6, w_5) \leftarrow & edge(w_8, w_9) \leftarrow \\ edge(w_1, w_{11}) \leftarrow & edge(w_9, w_8) \leftarrow \end{array}$$

Figure 4.1 shows a typical CHRONOLOG world graph which is generated by the user-defined operators *stack* and *unstack*. Edges are drawn in this graph by a special user-defined predicate *edge*.

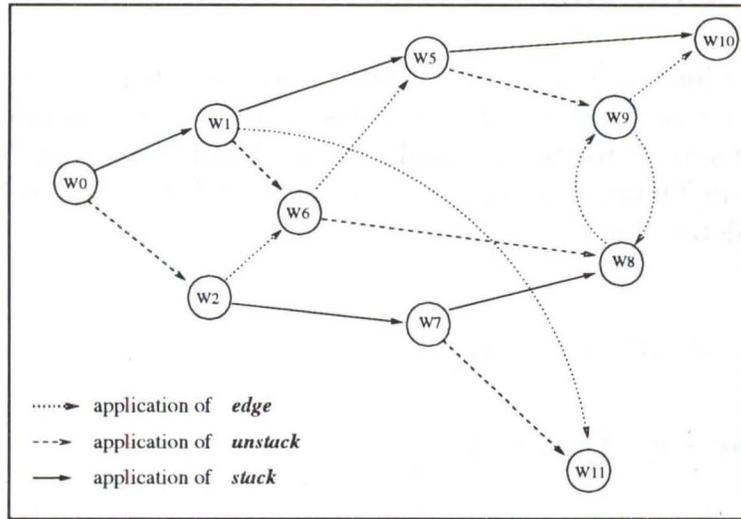


Figure 4.1: A CHRONOLOG world graph

4.1 The Syntax

We define the set $Term_{\Sigma, V}$ as the set of all *terms* with variables taken out of set V and function symbols out of Σ . Let Σ contain the special function symbols *true*, *edge* and *edge**. *true* is used for simplicity to represent an empty body of a clause. *edge* and *edge** are used by CHRONOLOGS built-in modal operators. For convenience we write variables starting with capital letters and all other symbols in lower case letters.

An *atom* is a term, that is not a variable. A *modality* is a nonempty sequence of terms.

CHRONOLOG clauses may be either an operator declaration (indicated by the neck symbol $:=$) or a conditional clause:

$$H := B$$

$$H \leftarrow B$$

Here H is a *goal*, that is of the form G_W and B is a *body goal* which may be one of:

- *true*
- G_W
- (B', B'')
- $\neg B'$
- $(\odot B')_W$

In all cases G is an atom, W a modality and B', B'' are bodygoals itself; \odot is an element of $\{\Box_f, \Diamond_f, \nabla_f, \Delta_f, \Box_p, \Diamond_p, \nabla_p, \Delta_p\}$. For convenience we write clauses of the form $H \leftarrow true$ as $H \leftarrow$

4.2 The Semantics

In [Sch89], an operational semantics for CHRONOLOG and a horizontal compilation scheme for the transformation of CHRONOLOG programs into PROLOG programs is defined. Since this scheme is very similar to the one used in [Pis91] and in chapter 7 for the knowledge representation system TRAM, we refrain from dealing with it here, and refer to the respective section for a more detailed description.

4.3 CHRONOLOG Examples

4.3.1 The Factory Example

Consider a factory where Mary, Joe and Tom work. They were hired long time ago and we don't know yet the exact time when they were hired. But we do remember that Mary and Tom worked before Tom, but not if Mary was hired before Tom. In CHRONOLOG we will write:

$$\begin{array}{ll} works(tom)_{w_0} \longleftarrow & edge(w_1, w_0) \longleftarrow \\ works(mary)_{w_1} \longleftarrow & edge(w_2, w_0) \longleftarrow \\ works(joe)_{w_2} \longleftarrow & \end{array}$$

We have the possibilities to hire new people and fire others. But we can only fire one if he already works, and we do not want to hire people that do already work and also not those that were fired some time ago or that disagree with someone who already works:

$$\begin{array}{l} fire(X)_W := works(X)_W \\ hire(X)_W := \neg(works(Y)_W, hates(Y, X)_W), \neg \diamond_p(works(X))_W \\ hates(john, tom)_W \\ hates(mary, paul)_W \\ hates(joe, jane)_W \end{array}$$

We can now define, that someone works at our factory if he was hired now or if he already worked the timepoint ago and was not fired now:

$$\begin{array}{l} works(X)_{hire(X)_W} \longleftarrow \\ works(X)_W \longleftarrow \Delta_p(works(X))_W, \neg eq(W, fire(X) \bullet W') \\ eq(X, X)_W \longleftarrow \end{array}$$

Now we can ask: if we have hired George and then Jane (after timepoint w_0), is there a future world in which Tom and Jane work in the factory, supposed that Paul has never worked in the meantime:

$\leftarrow eq(W, hire(susan) hire(george) w_0),$
 $works(tom)_{W'W},$
 $works(jane)_{W'W},$
 $\neg \diamond_p(works(paul))_{W'W}$

CHRONOLOG will compute the following modality

$$W' = hire(jane) fire(joe)$$

4.3.2 The blocks world

Let us now look a little bit closer at the blocks world example. We have several blocks, some of which lie on the table, whereas others are located on top of other blocks. We use six predicates to describe the specific *states* of our blocks world:

$on_table(X)$ means that block X is on the table
 $on(X, Y)$ means that block X is on block Y
 $clear(X)$ means that there is no other block on block X
 $holding(X)$ means that the robot arm is holding block X
 $armempty$ means that the robot arm is empty

Now in every state we can define four operators:

1. $stack(X, Y)$ can be applied in a certain world to two different blocks X and Y , if the roboter arm is holding X and there is no other block on Y .
2. $unstack(X, Y)$ is the inverse operation to $stack$: it can be applied, if X is on Y , the arm is empty and X is clear.
3. $pickup(X)$ can be applied, if the arm is empty and block X is free.
4. $putdown(X)$ is the last operator, that can be applied, if the arm is already holding block X .

The operator declarations will look in CHRONOLOG like:

$$\begin{array}{ll}
 stack(X, Y)_W := holding(X)_W, & unstack(X, Y)_W := on(X, Y)_W, \\
 & clear(Y)_W, \\
 & \neg eq(X, Y) \\
 & armempty_W
 \end{array}$$

$$\begin{array}{ll}
 pickup(X)_W := armempty_W, & putdown(X)_W := holding(X)_W \\
 & clear(Y)_W
 \end{array}$$

Now we can define several initial worlds, that represent certain states of our blocks world. First, in w_0 there are 3 blocks a, b and c that all lie on the table. The robot arm in this example world is empty:

$$\begin{aligned} \text{armempty}_{w_0} &\leftarrow \\ \text{on_table}(a)_{w_0} &\leftarrow \\ \text{on_table}(b)_{w_0} &\leftarrow \\ \text{on_table}(c)_{w_0} &\leftarrow \end{aligned}$$

In another world w_1 we have nearly the same situation, as in w_0 except that block a is already on block b :

$$\begin{aligned} \text{armempty}_{w_1} &\leftarrow \\ \text{on}(a, b)_{w_1} &\leftarrow \\ \text{on_table}(b)_{w_1} &\leftarrow \\ \text{on_table}(c)_{w_1} &\leftarrow \end{aligned}$$

Now we can declare clauses that specify the effects of applying the different operators:

$$\begin{aligned} \text{armempty}_{\text{stack}(X,Y) W} &\leftarrow \\ \text{on}(X, Y)_{\text{stack}(X,Y) W} &\leftarrow \\ \text{on}(U, V)_{\text{stack}(X,Y) W} &\leftarrow \text{on}(U, V)_W, \neg \text{eq}(U, X) \\ \text{on_table}(U)_{\text{stack}(X,Y) W} &\leftarrow \text{on_table}(U)_W, \neg \text{eq}(U, X) \end{aligned}$$

First we say that applying $\text{stack}(X, Y)$ leads to a world where the arm is empty. The second clause tells us that block X is on block Y after stacking. The last two clauses are frame axioms specifying that nothing else changes.

$\text{unstack}(X, Y)$ is defined similar to stack :

$$\begin{aligned} \text{holding}(X)_{\text{unstack}(X,Y) W} &\leftarrow \\ \text{on}(U, V)_{\text{unstack}(X,Y) W} &\leftarrow \text{on}(U, V)_W, \neg \text{eq}(U, X) \\ \text{on_table}(U)_{\text{unstack}(X,Y) W} &\leftarrow \text{on_table}(U)_W \end{aligned}$$

pickup and putdown are defined such that pickup can only be applied to blocks that lie on the table, and that putdown places a block always onto the table. Therefore, if we want to put the topmost block of a tower of blocks onto another tower of blocks, we first have to apply unstack (from the first tower) then putdown (on the table) then pickup (from the table) and lastly stack (onto the second tower). Therefore, the program looks like:

$$\begin{aligned} \text{armempty}_{\text{putdown}(X) W} &\leftarrow \\ \text{on_table}(X)_{\text{putdown}(X) W} &\leftarrow \\ \text{on_table}(Y)_{\text{putdown}(X) W} &\leftarrow \text{on_table}(Y)_W \\ \text{on}(Y, Z)_{\text{putdown}(X) W} &\leftarrow \text{on}(Y, Z)_W \end{aligned}$$

$$\begin{aligned}
& \text{holding}(X)_{\text{pickup}(X) W} \leftarrow \\
& \text{on_table}(Y)_{\text{pickup}(X) W} \leftarrow \text{on_table}(Y)_W, \neg \text{eq}(X, Y) \\
& \text{on}(Y, Z)_{\text{pickup}(X) W} \leftarrow \text{on}(Y, Z)_W, \neg \text{eq}(X, Y)
\end{aligned}$$

Using this program we can use different goals. For example:

$$\leftarrow (\nabla_p \text{on_table}(X))_{\text{stack}(a,b) \text{pickup}(a) w_0}$$

This goal asks (via backtracking) for all blocks that have always been on the table after having put a on b from world w_0 . (Here the answer will be first $X = b$ and after backtracking $X = c$).

Another goal asks how to build a tower, were a is on b and b on c , starting with w_1 :

$$\leftarrow \text{edge}^*(w_1, W), \text{on}(a, b)_W, \text{on}(b, c)_W$$

Here the correct answer should be:

$$W = \text{unstack}(a, b) \text{putdown}(a) \text{pickup}(b) \text{stack}(b, c) \text{pickup}(a) \text{stack}(a, b)$$

4.4 Conclusion

In this section, a new approach to incorporate time structures in PROLOG is described. The resulting CHRONOLOG system provides a set of modal operators to reason within a world graph that describes various states of a world in the future and the past. In addition the user can define new (conditioned) predicates which transform the objects and relations between the objects of the world to generate new possible worlds. Thus, CHRONOLOG combines and generalizes features from other temporal logic programming approaches; as in Tang's system it is possible just to operate on a predefined world graph or as in Abadi & Manna's system it is possible to work with operators that change the world status. In the following sections we will describe a temporal knowledge system based on the ideas of CHRONOLOG. The intended application domain for this is the simulation of multiagent environments, where planning, synchronization, and communication tasks have to be dealt with, strongly involving temporal aspects.

Chapter 5

The Basic Ideas of Tram

In the following sections, we describe the knowledge representation system TRAM which combines both interval and point aspects of time by integrating the ideas of two of the main paradigms for the representation of time in AI: Allen's interval logic [All84] and the modal logics approach [Pri67, AM89]. The former approach employs intervals and relations between intervals as the basic entities, whereas the latter one uses a set of modal operators which are interpreted by using a possible worlds semantics in order to express temporal knowledge. The foundations of our approach have been established by the CHRONOLOG system described in section 4. CHRONOLOG presents a temporal PROLOG based on standard modal logics. TRAM extends this concept by providing time intervals.

Our view of time is strongly driven by the requirements resulting from our research on multiagent-systems (MAS). In MAS, autonomous intelligent agents have to fulfil their own local goals in coordination with their environment and with other agents. This requires a great deal of synchronization, communication, and coordination work. Time is an essential concept for handling these kinds of tasks, since agents make their plans and decision within the flow of time. To provide a better idea of this, we will start by an example from a multi-agent scenario.

An Example

In this section we motivate the role of time and the way we handle it in our approach by a small example from a multi-agent domain. Figure 5.1 shows our exemplary scenario. There are two loading docks, d_1 and d_2 , two trucks t_1 and t_2 , and a railway line between loading station l_1 and l_2 . The two trucks receive the order to transport goods g_{11}, g_{12}, g_{21} , and g_{22} from the respective loading dock to loading station l_1 , to take the train to l_2 , and to unload there. Both trucks have to fetch goods both from l_1 and from l_2 . In this scenario time plays a role in a twofold manner: first, the two trucks have to synchronize their loading dock activities, since only one truck at a time can be served at each loading dock. Second, they have to coordinate their travel by train, since both trucks should take the same train to l_2 . An intelligent plan for t_1 and t_2 would be not to start at the same loading dock, but rather e.g. t_1 could first go to d_1 while t_2 could first go to d_2 , in order to avoid 'wait states'. Then

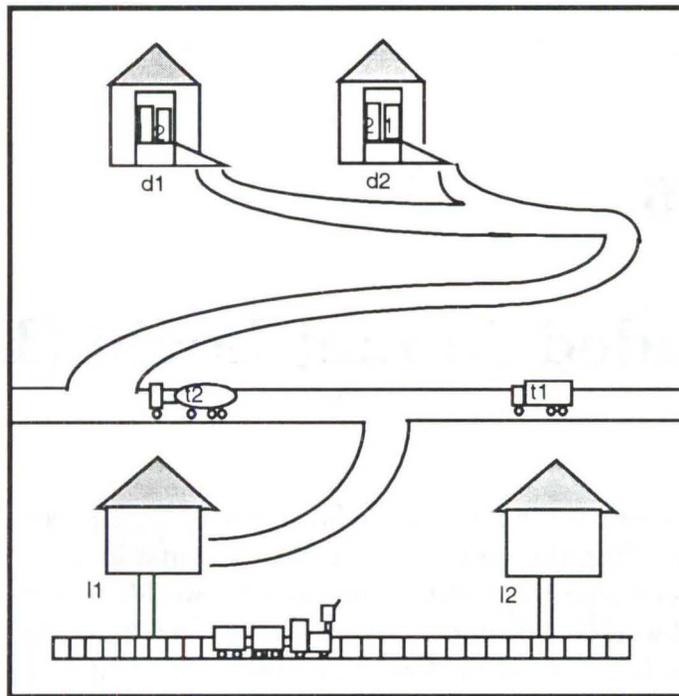


Figure 5.1: A Loading Dock Scenario

they could meet at l_1 and take the train to l_2 together. We suggest the use of time interval constraints in order to synchronize the actions of t_1 and t_2 . A formal solution to the example using the TRAM system is described in chapter 7. The complete TRAM program modelling the problem can be found in appendix A.

Chapter 6

An Extended Modal Logic (EML)

In this chapter, we present the extended modal logics EML. In section 6.1, we define the syntax of EML by extending the syntax of a standard modal logics by additional temporal operators, and by generalizing the notion of modal logic worlds to *time instances*, which are defined as the union of worlds and time intervals. In section 6.2 we provide a model-theoretic semantics of EML. Finally, in section 6.3 we show how to draw inferences over time intervals. For reasons of space, we assume that the reader be familiar with FOPL and modal logics. We presuppose notions such as term, wff, interpretation, model, satisfiability, tautology etc. (cf. [BB87] for an introduction into FOPL and [Ram88] for modal logics).

6.1 The Syntax of EML

The basic primitives of the time model underlying to EML are *worlds*. *Intervals* are defined as closed sequences of worlds. Starting from this we can now define *time instances*.

Definition 6.1.1 *Be \mathcal{W} a non-empty set of worlds. The relation \sqsupset defines a partial order on elements of \mathcal{W} (accessibility relation). We define the set \mathcal{I} of intervals as:*

$$\mathcal{I} := \{ \langle w_0, \dots, w_n \rangle \mid \forall i : 0 \leq i \leq n \quad w_i \in \mathcal{W} \wedge \forall k : 0 \leq k < n \quad w_k \sqsupset w_{k+1} \}$$

The set \mathcal{T} of time instances is $\mathcal{T} := \mathcal{W} \cup \mathcal{I}$.

Next, we define the *accessibility relation* \sqsupset^+ over arbitrary elements of \mathcal{T} in the frame system $F = (\mathcal{T}, \sqsupset^+)$.

Definition 6.1.2 *Be \mathcal{T} a non-empty set of time instances, and be \sqsupset^+ a binary relation on elements of \mathcal{T} . For arbitrary elements $t_1, t_2 \in \mathcal{T}$, $t_1 \sqsupset^+ t t_2$ holds iff:*

1. $t_1, t_2 \in \mathcal{W}$ and $t_1 \sqsupset t_2$
2. $t_1 \in \mathcal{W}, t_2 \in \mathcal{I}$, with $t_2 \equiv \langle w_0, \dots, w_n \rangle$ and $t_1 \sqsupset w_0$

3. $t_1 \in \mathcal{I}, t_2 \in \mathcal{W}$, with $t_1 \equiv \langle w_0, \dots, w_n \rangle$ and $w_n \sqsupset t_2$

4. $t_1, t_2 \in \mathcal{I}$, with $t_1 \equiv \langle w_{1_0}, \dots, w_{1_n} \rangle$, $t_2 \equiv \langle w_{2_0}, \dots, w_{2_k} \rangle$ and $w_{1_n} \sqsupset w_{2_0}$

In order to be able to describe both worlds and intervals, we introduce new modal operators such as $\triangle, \nabla, \blacklozenge, \blacksquare, \diamond$ and \boxminus .

Definition 6.1.3 (EML) *The extended modal logic language EML is defined by adding to a standard modal logic language ML the logical symbols $\nabla, \triangle, \blacklozenge, \blacksquare, \diamond$ und \boxminus . Be Φ a well-formed formula (wff) in ML, and $i \equiv \langle w_0, \dots, w_n \rangle$ in \mathcal{T} , then $\Phi, \triangle\Phi, \nabla\Phi, \diamond\Phi, \square\Phi, \blacklozenge\Phi, \blacksquare\Phi, \diamond\Phi$ and $\boxminus\Phi$ are wff's in EML. They are to be read as follows:*

- $\triangle\Phi$: There exists an immediately successing time instance, at which Φ holds.
- $\diamond\Phi$: There exists a time instance at which Φ holds.
- $\blacklozenge\Phi$: There exists a time interval during which Φ holds.
- $\diamond\Phi$: In the (current) time interval i there exists a time instance at which Φ holds.

Analogously, we define the symbols $\nabla, \square, \blacksquare$ und \boxminus for universal quantification of time instances and intervals, respectively.

In the next section, we provide a semantics for EML terms and formulas.

6.2 A Model-Theoretic Semantics of EML

We define the semantics of EML in terms of Kripke structures and Kripke interpretations, which is a quite familiar technique for modal logic approaches. In the following, we write “ $<$ ” for the *accessibility relation* \sqsupset^+ over \mathcal{T} whenever it becomes clear from the context what is meant.

Definition 6.2.1 *A Kripke interpretation is a tuple $M = (\mathcal{T}, <, \mathcal{D}, I)$, where the following holds:*

1. \mathcal{T} is a non-empty set of time-instances. \mathcal{W} be the set of worlds in \mathcal{T} .
2. $<$ is a (partial) binary relation on \mathcal{T} .
3. \mathcal{D} is a non-empty set of individuals.
4. I is a function which maps each n -ary function symbol $f \in \mathcal{F}$ to an n -ary function f^M on \mathcal{D} and each n -ary predicate symbol $p \in \mathcal{P}$ in each world $w \in \mathcal{W}$ to an n -ary relation p_w^M on \mathcal{D} , so that the following holds:

$$(a) p_w^M \subseteq \mathcal{D}^n, \text{ if } p \in P^f \text{ and}$$

(b) $p_w^M = p_{w'}^M$ for all $w, w' \in \mathcal{W}$, if $p \in P^r$.

Note that I defines a standard interpretation function for modal logics. Since worlds are the basic primitives of the language, I is only defined over worlds. Interval formulas are interpreted by pulling them down to the worlds contained in the interval. As usual, function symbols have a *fixed* interpretation whereas predicate symbols are *flexibly* interpreted¹. Next we define how terms and formulas are interpreted. As the reader will see this is quite similar to standard modal logics.

Definition 6.2.2 *Be \mathcal{V} the set of all individual variables, and be \mathcal{D} the set of all individual constants. A **variable assignment** relating to a Kripke interpretation $M = (\mathcal{T}, <, \mathcal{D}, I)$ is a mapping $\alpha: \mathcal{V} \rightarrow \mathcal{D}$. The value $\alpha_w(x)$ of a term x in a world $w \in \mathcal{T}$ is defined as follows:*

1. $\alpha_w(x) = \alpha(x)$ for $x \in \mathcal{V}$.
2. $\alpha_w(f(t_1, \dots, t_n)) = I(f)(\alpha_w(t_1), \dots, \alpha_w(t_n))$, otherwise.

Here we consider only *global* variables², i. e. the value of a term does not depend on the actual world.

Definition 6.2.3 *Be $M = (\mathcal{T}, <, \mathcal{D}, I)$ an interpretation, α an assignment function. M, α satisfy the formula Φ in a time instance $t_0 \in \mathcal{T}$ if the following holds true:*

1. Φ is an atomic formula $P(t_1, \dots, t_n)$, and $I_{t_0}(P)(\alpha_{t_0}(t_1), \dots, \alpha_{t_0}(t_n))$ holds (abbrev. $(M, \alpha) \models_{t_0} P(t_1, \dots, t_n)$).
2. Φ is $\neg\Phi_1$, and $(M, \alpha) \models_{t_0} \Phi_1$ does not hold.
3. Φ is $\Phi_1 \rightarrow \Phi_2$ and $(M, \alpha) \models_{t_0} \Phi_1$, then also $(M, \alpha) \models_{t_0} \Phi_2$.
4. Φ is $\Delta\Phi_1$ and $t' \in \mathcal{T}$ exists with $t_0 < t'$: $(M, \alpha) \models_{t'} \Phi_1$ and there exists no $t'' \in \mathcal{T}$ with $t_0 < t'' < t'$.
5. Φ is $\Diamond\Phi_1$ and $(M, \alpha) \models_{t_0} \Phi_1$ or a $t' \in \mathcal{T}$ exists with $t_0 < t'$, such that $(M, \alpha) \models_{t'} \Phi_1$.
6. Φ is $\blacklozenge\Phi_1$ and $(M, \alpha) \models_{t_0} \Phi_1$ for $t_0 \in \mathcal{I}$, or an $i \in \mathcal{I}$ exists with $t_0 < i$, such that $(M, \alpha) \models_i \Phi_1$.
7. Φ is $\lozenge\Phi_1$ and $t_0 \in \mathcal{I}$, with $t_0 \equiv \langle w_0, \dots, w_n \rangle$ and a $w \in t_0$ exists, such that $(M, \alpha) \models_w \Phi_1$.
8. Φ is $\xi\Phi_1$ with $\xi \in \{\nabla, \square, \blacksquare, \boxplus\}$, and $(M, \alpha) \models_{t_0} \neg\xi'\neg\Phi_1$, where ξ' is the corresponding element to ξ of $\{\Delta, \Diamond, \blacklozenge, \lozenge\}$.

¹If we want to allow both fixed and flexible function symbols, the definition of I must be extended (cf. [Brz89]).

²If we would like to consider local variables as well, the assignment function α has to be modified, respectively.

The notions of *semantic consequence* and *tautology* are defined exactly as in classical modal logics. In the following, we will provide some examples in order to give an idea of the expressiveness of a formalism for knowledge representation based on EML.

Example 6.2.4 Classical modal logics are typically restricted to inferences of the type 'Does a world w exist in which a certain goal G holds?'. So we could ask if there exists a world in which the loading order of truck $truck1$ is empty. By using EML, more expressive inferences are possible:

1. There exists a time interval in which $truck1$ goes by train and in which its loading order is always empty:

- $\models_t \blacklozenge(\exists(\text{gobyTrain}(truck1) \wedge \text{loadingOrder}(truck1, [])))$

2. There exists an interval i during which truck $truck1$ is dispatched at loading dock $ramp1$, and there exists no interval following directly to i in which $truck1$ stays at $ramp1$:

- $\models_t \blacklozenge(\lozenge(\text{atRamp}(truck1, ramp1)) \wedge \nabla \neg \lozenge(\text{atRamp}(truck1, ramp1)))$

6.3 Integrating Interval Logic

In this section we describe how Allen's interval relations can be embedded into EML. We achieve this by means of a binary function **Access** which is globally defined. Its value at an arbitrary time instance t is a special **time value** associated to t . For $t \in \mathcal{I}$ the time value consists of the first and last point of the interval, for $t \in \mathcal{W}$ it consists of two identical values. Intuitively, the time value is an abstract measure, which can be modelled e. g. by a real number or by an integer. *Access* allows to compare arbitrary worlds as regards the temporal relationship between these worlds, even if the accessibility relation is only partial. In the implementation of TRAM, a length value is assigned to each edge in the graph of worlds, and the time value of a world computes as the sum of the edges starting from the current world. Now, we formally extend the notion of interpretation by the *Access* function:

Definition 6.3.1 *An interpretation is a tuple $M' = ((\mathcal{T}, <, \mathcal{D}, I), \text{Access})$ where $(\mathcal{T}, <, \mathcal{D}, I)$ corresponds to the interpretation of definition 6.2.1, and the function **Access**: $\mathcal{V}^2 \rightarrow \mathcal{N}^2$ defines the access to a time instance.*

Now, we are able to define the thirteen Allen relations as binary predicates over the time values. The transformation of the relation into a representation based on instants rather than intervals, and the axiomatization for the set of natural numbers do not cause serious problems. Therefore, we desist from a more detailed description of this issue and refer to literature (e. g. [Ram88]). Rathermore, we provide some examples taken from our loading-dock scenario to illustrate the usefulness of EML:

In the following examples we show how several patterns of synchronization of resources between agents can be represented by using EML.

1. The first example shows how the synchronization of shared resources can be represented using EML. Assume that two trucks $truck_1$ and $truck_2$ have to meet at the loading-station $station_1$ in order to take the train together. So we can ask whether there is a time interval in the future (starting from time t) where the trucks $truck1$ and $truck2$ are both at the station? (This is the precondition for them to use the shared resource *train*.)

$$\begin{aligned} \models_t & \blacklozenge(\text{at_station}(\text{truck1}, \text{station1})) \quad \wedge \text{Access}(\text{X1}, \text{Y1}) \wedge \\ & \blacklozenge(\text{at_station}(\text{truck2}, \text{station1})) \quad \wedge \text{Access}(\text{X2}, \text{Y2}) \wedge \\ & \text{equals}(\text{X1}, \text{Y1}, \text{X2}, \text{Y2}) \end{aligned}$$

2. In many applications agents which independently pursue their own goals have access to common resources which may only be used exclusively. This access has to be synchronized. In our loading-dock example the loading ramps can be considered as exclusive resources, since at most one agent (truck) is allowed to stay at a ramp $ramp1$ at time t . So we could ask in EML, whether a world exists starting from time t in which both $truck1$ and $truck2$ have achieved their goals and the access to the ramp $ramp1$ has been scheduled:

$$\begin{aligned} \models_t & \blacklozenge_f(\text{goal_achieved}(\text{truck1}) \wedge \blacklozenge_p(\text{at_ramp}(\text{truck1}, \text{ramp1}) \wedge \text{Access}(\text{X1}, \text{Y1}))) \wedge \\ & \blacklozenge_f(\text{goal_achieved}(\text{truck2}) \wedge \blacklozenge_p(\text{at_ramp}(\text{truck2}, \text{ramp1}) \wedge \text{Access}(\text{X2}, \text{Y2}))) \wedge \\ & (\text{after}(\text{X1}, \text{Y1}, \text{X2}, \text{Y2}) \vee \text{before}(\text{X1}, \text{Y1}, \text{X2}, \text{Y2})) \end{aligned}$$

Chapter 7

The Knowledge Representation System Tram

In this section we develop a computational model of EML which is based on PROLOG. We presume that the reader be familiar with this language and with standard logic programming in general (cf. [CM81, SS86] for PROLOG [Llo84] for logic programming). After giving a more informal idea of how TRAM works, we will specify an operational semantics for TRAM 7.1 and a compilation scheme for TRAM programs into PROLOG programs 7.2. Section 7.3 recalls our loading-dock scenario presented in chapter 5 and outlines a solution to it.

The implementation of modal logic propositions is coupled with the representation of different worlds and of properties associated to these worlds. In TRAM we describe those worlds by constant declarations w_0, w_1 etc. together with a unary predicate $world/1$. The properties (valid propositions) of a world w_i are represented by PROLOG clauses. Propositions which are valid in any world are bound to a variable world name.

The modal logic *accessibility relation* defines a partial order on the graph of worlds, i. e. it defines which worlds are reachable (in the *future* or *past*) from a given world. We express this in TRAM by defining edges between worlds using a predicate $edge/2$. Apart from considering worlds that actually exist, TRAM allows us to *compute* possible worlds starting from the current world. For this purpose, we define transition operators with *preconditions* and *postconditions*, which allow transitions between worlds if their preconditions are satisfied. A world which has been computed this way is identified by the list of operators applied successively beginning from the current world. In order to be able to decide which predicates hold in computed worlds, the effects of the application of a transition operator must be defined by its postcondition. We can define both *primitive* transition operators (POs) and *macro* transition operators (MO). MOs can be constructed as sequences of POs, and they can be arbitrarily nested. Since MOs define sequences of worlds, representing an interval by an MO is a very natural idea.

Note, that expanding worlds by using operators does not cause new facts to be added to the knowledge base. Rathermore, since the computation of a world w_i is decoded into the world name as the path of applied operators, the truth of formulas in w_i can be computed by the

name of w_i ¹.

For formulating queries, TRAM provides predicates such as `f_diamond_trans/2`, whose first argument is a goal clause, and whose second argument is bound to the name of a world for which the query could be proved (if such a world exists).

At the end of this informal motivation, we should mention how propositions depending on specific worlds are actually handled in TRAM. Each clause of a TRAM program is translated into a PROLOG clause by using a binary function *reify*, which adds to each clause an argument representing the world the clause refers to. This will become more apparent in section 7.2 where we define a compilation scheme for TRAM.

7.1 An Operational Semantics of TRAM

In this section we present a scheme of computation for our knowledge representation language TRAM which is based on the extended modal logic EML we introduced in chapter 6. Since we intend a modal logic representation, world indices W are attached to goals in order to express that the evaluation of a program depends on the current world W . In TRAM a distinction is made between two kinds of program clauses: the first one declares transition operators modifying the world and is represented by ' $H := B$ ', the second one defines the usual program clauses which associate propositions to worlds. It is represented ' $H \leftarrow B$ ' as we know it from PROLOG. H is a Goal G_W , B is a body goal which can be either $\{true, G_W, (B', B''), \neg B', (\odot B')_W\}$. G represents an atom, W a world name, B', B'' are body goals, and \odot stands for an arbitrary element of $\{\Delta_f, \nabla_f, \square_f, \diamond_f, \blacklozenge_f, \blacksquare_f, \boxplus, \boxtimes, \Delta_p, \nabla_p, \square_p, \diamond_p, \blacklozenge_p, \blacksquare_p\}$. For the sake of simplicity we write $H \leftarrow$ as an abbreviation of $H \leftarrow true$.

Central notions for the operational semantics we provide in the following are *substitution*, *unifier*, and *most general unifier (mgu)*. They are defined exactly as it is the case in PROLOG. Since world arguments are represented by PROLOG data structures (i. e. lists), it makes sense to talk about the mgu of two worlds, and this mgu (if it exists) can be computed by the standard PROLOG unification algorithm. Now, we define the operational semantics of a TRAM program.

Definition 7.1.1 (Operational semantics of TRAM) *Be P a TRAM program, G a goal, σ a substitution. G is a logical consequence of P under σ ($P \models_\sigma G$) if $P \cup G$ is contradictory under σ . The following cases must be considered:*

1. $G = true$, with σ is the empty substitution
2. $G = B_W$, with $H'_{W'} \leftarrow B' \in P$
and $\sigma' = mgu(B_W, H'_{W'})$
and $P \models \sigma'(B')$ under σ

¹The implicit representation of applications of operators establishes our solution of the *frame-problem*

3. $G = (B', B'')$, with $P \models B'$ under σ'
and $P \models \sigma'(B'')$ under σ
4. $G = (\neg B)$, with there ex. no $\sigma' : P \models B$ under σ'
and σ is the empty substitution
5. $G = (\odot_f B)_W$, with $P \models \neg(\otimes_f \neg B)_W$ under σ
6. $G = (\diamond B)_W$, with $P \models B_W$ under σ
or
 $MOp_{W''} := B' \in P$
and $W = \sigma'(MOpW')$
and $\sigma' = mgu(W'', W')$
and $P \models \text{prove}^*(B, \sigma'(W'), \sigma'(B'))$ under σ
7. $G = (\triangle_f B)_W$, with $P \models \text{edge}(W, W')$ under σ'
and $P \models \sigma'(B_{W'})$ under σ
8. $G = (\diamond_f B)_W$, with $P \models \text{edge}^*(W, W')$ under σ'
and $P \models \sigma'(B_{W'})$ under σ
9. $G = (\blacklozenge_f B)_W$, with $P \models \text{edge}^*(W, W')$ under σ'
and $P \models \sigma'(B_{W'})$ under σ
10. $G = \text{edge}^*(W, W')$, with $\sigma = mgu(W, W')$
11. $G = \text{edge}^*(W, W')$, with $P \models \text{edge}(W, W'')$ under σ'
and $P \models \text{edge}^*(\sigma'(W''), \sigma'(W'))$ under σ
12. $G = \text{edge}(W, W')$, with $Op_{W''} := B \in P$
and $\sigma' = mgu(W, W'')$
and $P \models \sigma'(B)$ under σ
and $W' = \sigma'(OpW)$
13. $G = \text{prove}^*(B, W, (B', B''))$
 , with $P \models \text{prove}(B, W, B')$ under σ
or $P \models \text{prove}^*(B, W, B'')$ under σ
14. $G = \text{prove}(B, W, Op_{W''})$
 , with $\text{edge}(W, W')$ under σ'
and $W' = \sigma''(\sigma'(Op_{W''}))$
and $P \models \sigma''(\sigma'(B_{W'}))$ under σ

\odot stands for an arbitrary element of the set $\{\nabla_f, \square_f, \blacksquare_f, \boxplus\}$, \otimes stands for the element corresponding to \odot of $\{\triangle_f, \diamond_f, \blacklozenge_f, \diamond\}$. P represents the current set of TRAM clauses, W, W', W'' , and W''' are world names (world arguments), G is a goal, B, B' , and B'' are body goals. Op_W and MOp_W are clause headers declaring primitive and macro operators, respectively. OpW (resp. $MOpW$) denotes concatenation. Intuitively, it means that the

world named OpW ($OpMW$) is accessed starting from a world W using the operator Op . The modal operators $\{\nabla_p, \square_p, \blacksquare_p, \Delta_p, \diamond_p, \blacklozenge_p\}$, which allow statements about the past can be defined analogously.

Note that 1) ... 4) correspond to PROLOG SLD resolution. The index W in case 2) merely expresses that the clause depends on an additional world argument. Case 5) treats the modal-logic rules of double negation for *box* operators. Cases 6) ... 9) maintain the *diamond* operators using the meta-predicates *edge*, *edge**, *prove*, and *prove**. In cases 10) ... 14) the semantics of the meta-predicates *edge*, *edge**, *prove*, and *prove** is defined. *edge*(W, W') specifies an edge connecting two worlds W and W' in the graph of worlds. The predicate *prove*(G, W, W') succeeds if the goal G can be proved in a world W' starting from the current world W .

7.2 A Horizontal Compilation Scheme for TRAM Programs

In this section we provide a formal scheme for compiling TRAM clauses into PROLOG. The basic idea is to transform each n -ary TRAM goal $p(t_1, \dots, t_n)_W$ into an $(n+1)$ -ary PROLOG goal $p(t_1, \dots, t_n, W)$, whose last argument W represents the modality, i. e. the world. The compilation itself is performed by the procedure C . The function *reify* defines the way a single TRAM-goal is transformed:

Definition 7.2.1 *The function reify maps each TRAM goal $p(t_1, \dots, t_n)_{W_1, \dots, W_k}$ to a PROLOG goal $p(t_1, \dots, t_n, W_1 \bullet \dots \bullet W_k)$ as follows:*

$$\text{reify}(p(t_1, \dots, t_n)_{W_1, \dots, W_k}) := p(t_1, \dots, t_n, W_1 \bullet \dots \bullet W_k).$$

Here, the function symbol \bullet symbolizes the concatenation of world arguments.

Example 7.2.2 Be Q_{W_1} the goal $on(a, b)_{w_0}$ and R_{W_2} the goal $on_table(X)_{putdown(X), W}$. *reify* translates Q_{W_1} and R_{W_2} as follows:

$$\text{reify}(on(a, b)) \Rightarrow on(a, b, w_0).$$

$$\text{reify}(on_table(X)_{putdown(X), W}) \Rightarrow on_table(X, putdown(X) \bullet W).$$

In the current implementation the world argument is represented as a list containing the start world and the sequence of operator application. ♠

In the following we explain how a TRAM program is translated into a PROLOG program. The procedures C , C' and C'' actually define the horizontal compilation scheme for TRAM programs. In C a distinction is made between operator declarations and 'normal' program clauses. C' and C'' perform the translation of the different goals.

Definition 7.2.3 The procedure C maps the set of TRAM clauses TCL to the set of horn clauses HCL . $C : TCL \rightarrow HCL$ is defined as follows:

$$C[cl] := \begin{cases} op(POp, W') \leftarrow C'[B] & \text{for } cl = (POp_W := B) \\ op(reify(MOp_{W'}), W) \leftarrow C''[B, W, W'] & \text{for } cl = (MOp_W := B) \\ C'[H] \leftarrow C'[B] & \text{for } cl = (H \leftarrow B) \end{cases}$$

W is a new variable, POp is a primitive operator declaration, and MOp is a macro operator declaration.

The first case in the above definition handles the declaration of a primitive operator. The corresponding body B is processed by the procedure C' . B consists of conditions that have to be satisfied in a world where the operator shall be applied. In the case of declarations of macro operators (case 2) also primitive operators can be applied. Therefore, such a body is translated by a special procedure C'' .

Definition 7.2.4 The auxiliary functions C' and C'' map each body goal B to a new body goal HB . C' is defined as follows:

$$C'[B] := \begin{cases} true & \text{for } B = true \\ reify(B) & \text{for } B = G_W, \quad G \text{ term} \\ C'[B'], C''[B''] & \text{for } B = (B', B'') \\ \neg C'[B'] & \text{for } B = \neg B' \\ \neg C'[(\otimes_f \neg B')_W] & \text{for } B = (\odot_f B')_W \\ edge(W, W'), C'[B'_{W'}] & \text{for } B = (\triangle_f B')_W \\ edge(W', W), C'[B'_{W'}] & \text{for } B = (\triangle_p B')_W \\ edge^*(W, W'), C'[B'_{W'}] & \text{for } B = (\diamond_f B')_W \\ edge^*(W', W), C'[B'_{W'}] & \text{for } B = (\diamond_p B')_W \\ edge^*(W, W'), C'[B'_{W'}] & \text{for } B = (\blacklozenge_f B')_W \\ edge^*(W', W), C'[B'_{W'}] & \text{for } B = (\blacklozenge_p B')_W \\ (C'[B'_{W'}]; & \text{for } B = (\diamond B')_W \\ (W = MOp \bullet W', & \text{and there ex. } (MOp \leftarrow Mbody) \\ C'[B'_{W_1}], & \\ edge^*(W', W_1), & \\ edge^*(W_1, Mbody \bullet W')) & \end{cases}$$

In the above definition, \odot stands for an arbitrary element of the set $\{\nabla_f, \square_f, \blacksquare_f, \boxplus\}$, \otimes stands for the element corresponding to \odot of $\{\triangle_f, \diamond_f, \blacklozenge_f, \diamond\}$.

If B is an atomar body goal $B = G_W$, the corresponding world argument is simply appended by means of the function $reify$. If the body goal has the form $B = (\triangle_f B')_W$, the existence of exactly one edge ($edge(W, W')$) in the graph of worlds directed to the future is required, and the corresponding goal $B'_{W'}$ is processed in the new world. For $B = (\diamond_f B')_W$, the existence of at least one edge pointing to the future is required. If $B = (\diamond B')_W$, W is expected to be

an interval. There are two possibilities to prove $B'_{W'}$. Firstly, $B'_{W'}$ may ensue directly from applying a macro as a normal transformation operator, and thus follow from the current world. Secondly, it can follow from one of the worlds contained in the interval.

The function C'' provides the translation of macro operators and their corresponding body goals. In the body of a macro declaration we make a distinction between calls to primitive operators, which have to be translated separately, and other conditions which can be processed by C' itself:

$$C''[B, W, W_b] := \begin{cases} C''[B_1, W, W_1], C''[B_2, W_1, W_b] & \text{for } B = (B_1, B_2) \\ op(POp, W), W_b = POp \bullet W & \text{for } B = POp \\ C'[B_W], W_b = W & \text{otherwise} \end{cases}$$

Finally, we will spend a few words on the actual TRAM run-time system. It solely consists of two predicates $edge$ and $edge^*$:

- $edge^*(W, W) \leftarrow$
 $edge^*(W, W') \leftarrow edge(W, W''), edge^*(W'', W')$
- $edge(W, Op \bullet W) \leftarrow op(Op, W)$

Intuitively, $edge(W, W')$ finds an edge from one world W to a next possible world W' , if such an edge exists. This depends on whether there is an operator declaration whose operator Op can be applied. Then the operator and the world argument are concatenated to the new world argument $Op \bullet W$. The first case of the specification of $edge^*$ is necessary because of the reflexivity of the accessibility relation, i. e. since the current world is always a possible world.

7.3 Recalling the Loading Dock Scenario

This section contains excerpts of a TRAM program representing the loading-dock scenario basically defined in section 5. Figure 7.1 contains some parts of the corresponding TRAM database. Some definitions of primitive and macro operators with pre- and postconditions are shown. The primitive operator $moveToRamp(Agent, Ramp)$ can be applied if the Agent has to load or to unload something at the ramp, if it has not just moved back from the ramp in the previous world (this is to avoid trivial circularities), and if it has already entered the loading-dock area, but is not yet at the ramp. After performing move to ramp, in the new world, it is true that the Agent is at the ramp and can now perform its loading or unloading job.

A crucial concept is that of macro operators, which can be defined as compositions of simple operators. Thus, macro operator $getGoods$ is defined by first moving to the ramp, then loading, and finally moving back from the ramp. Thus, the macro operator defines an interval consisting of a starting world, two intermediate worlds defined by applying $moveToRamp$ to the starting world and by applying $load$ to this world, respectively, and of a final world

```

%*****
% Loading Dock Scenario: Source Code
%*****

%***** Primitive Operators movetoRamp, load, moveBack *****

%-----
% Preconditions for a truck moving to a loading dock

prim_op( moveToRamp( Agent, LoadingDock)):-
  isAgent( Agent ), isRamp( LoadingDock),
  not done( moveBack( Agent, LoadingDock)),
  entered( Agent, LoadingDock ), not atRamp( _, LoadingDock),
  hasToLoadAtRamp( Agent, LoadingDock ).
%-----
% Preconditions for a truck being loaded

prim_op( load( Agent, LoadingDock )) :-
  isAgent( Agent), isRamp( LoadingDock ),
  atRamp( Agent, LoadingDock ), hasToLoadAtRamp( Agent, LoadingDock ).
%-----
% Preconditions for a truck moving away from a loading-dock

prim_op( moveBack( Agent, LoadingDock)):-
  isAgent( Agent ), isRamp( LoadingDock),
  not done( moveToRamp( Agent, LoadingDock)), atRamp( Agent, LoadingDock ).

%***** Macro operators *****

macro_op( getGoods( Agent, LoadingDock )) :-
  moveToRamp( Agent, LoadingDock),
  load( Agent, LoadingDock),
  moveBack( Agent, LoadingDock ).

%*** Postconditions of the operators movetoRamp, load, moveBack, getGoods ***

moveToRamp( Agent, LoadingDock).done( moveToRamp( Agent, LoadingDock)).
moveToRamp( Agent, LoadingDock).atRamp( Agent, LoadingDock ).
moveToRamp( Agent, LoadingDock).loadingOrder( A, 0 ):-
  loadingOrder( A, 0 ).

load( Agent, LoadingDock ).atRamp( Agent, LoadingDock ).
load( Agent, LoadingDock ).loadingOrder( Agent, NewOrder ):-
  loadingOrder( Agent, OldOrder),
  {delete( OldOrder, (LoadingDock,Good), NewOrder)}.

moveBack( Agent, LoadingDock).isDriving( Agent ).
moveBack( Agent, LoadingDock).done( moveBack( Agent, LoadingDock)).
moveBack( Agent, LoadingDock).loadingOrder( A, 0 ):-
  loadingOrder( A, 0 ).

getGoods( Agent, LoadingDock).isDriving( Agent ).
getGoods( Agent, LoadingDock).loadingOrder( Agent, NewOrder ):-
  loadingOrder( Agent, OldOrder),
  {delete( OldOrder, (LoadingDock,Good), NewOrder)}. %% ...

:- prolog.
thisWorld(W,W).

```

Figure 7.1: A TRAM database for the Loading Dock Scenario

after applying the macro. By mapping time values (durations) to the single actions, we can formulate how much time has passed by going from the start world to the end world of the interval.

In figure 7.2, a possible start world for the loading dock scenario is described. In this start world, both trucks *truck1* and *truck2* are driving and have orders to accomplish. Moreover, there are some propositions true in any world (expressed by the variable world name *EveryWorld*. E. g. both trucks are agents, for both trucks *station2* is the target station, and the only way to reach *station2* from *station1* is by taking the train. By the way, it seems the only way to reach *station2* at all.

```

%***** WORLD PREDICATES ... *****

:- world( startWorld ).
isDriving( truck1 ).
loadingOrder( truck1, [( loadingdock1, g11 ), ( loadingdock2, g21 ) ] ).
isDriving( truck2 ).
loadingOrder( truck2, [( loadingdock1, g12 ), ( loadingdock2, g22 ) ] ).

:- world( EveryWorld ).
isAgent( truck1 ).
hasToGoByTrain( truck1, station1, station2 ).
isTargetStation( truck1, station2 ).

isAgent( truck2 ).
hasToGoByTrain( truck2, station1, station2 ).
isTargetStation( truck2, station2 ).

isRamp( loadingdock1 ).
isRamp( loadingdock2 ).
isTrainRamp( station1 ).
isTrainRamp( station2 ).

hasToLoadatRamp( Agent, LoadingDock ) :-
  isAgent( Agent ), isRamp( LoadingDock ), loadingOrder( Agent, List ),
  {member( (LoadingDock,Good), List )}. %%% This goal invokes original PROLOG

```

Figure 7.2: A TRAM Starting World

Finally, in figure 7.3, we provide a query to the TRAM system delivering a world in which a task schedule has been achieved where 'wait states' of trucks at ramps are avoided and the two trucks finally meet at the station to board the train. Note that the variables *Res1* and *Res2* incorporate the 'plans' for *truck1* and *truck2*, respectively, i. e. the sequence of actions which transmit them into a world in which their goals are fulfilled. A listing of the complete solution to the problem can be found in Appendix A.

```

% TRAM query ensuring a scheduling of the two trucks. The predicates
% f_diamond_trans and p_full_diamond_trans correspond to the EML operators
% defined previously.

?- f_diamond_trans( orderDelete(truck1), Res1 , startWorld),
   f_diamond_trans( orderDelete(truck2), Res2 , startWorld),
   not( p_full_diamond_trans(
        (thisWorld( C1 ), { C1 =[getGoods(truck1,RampI,_)|_],
          p_full_diamond_trans( (thisWorld( C2 ),
            { C2=[getGoods(truck2,RampI,_)|_] } ),_,Res2),
          C1 equal C2} ),_,Res1)
      ).

%***** Answer Variable Bindings *****

Res1 =
  unload(truck1).goByTrain(truck1,rampT2, [useTrain(truck1,rampT1,rampT2),changeToTrain(truck1,rampT1)])
  .getGoods(truck1,ramp1, [moveBack(truck1,ramp1),load(truck1,ramp1),moveToRamp(truck1,ramp1)])
  driveTo(truck1,ramp1)
  .getGoods(truck1,ramp2, [moveBack(truck1,ramp2),load(truck1,ramp2),moveToRamp(truck1,ramp2)])
  driveTo(truck1,ramp2).startWorld,

Res2 =
  unload(truck2).goByTrain(truck2,rampT2, [useTrain(truck2,rampT1,rampT2),changeToTrain(truck2,rampT1)])
  .getGoods(truck2,ramp2, [moveBack(truck2,ramp2),load(truck2,ramp2),moveToRamp(truck2,ramp2)])
  driveTo(truck2,ramp2)
  .getGoods(truck2,ramp1, [moveBack(truck2,ramp1),load(truck2,ramp1),moveToRamp(truck2,ramp1)])
  driveTo(truck2,ramp1).startWorld,

C1 = _161, RampI = _205, C2 = _258

```

Figure 7.3: A Query to the TRAM System

Chapter 8

Conclusion and Outlook

Conclusion

In this paper we have tried to provide an overview on some current existing approaches on the representation of temporal knowledge. After presenting some important work on this subject, we described the CHRONOLOG temporal programming language and the TRAM system for the representation of temporal knowledge based on CHRONOLOG. TRAM integrates two different concepts of time: one based on modal logics and another one related to intervals of time. We have shown how intervals can be represented as connected sequences of worlds in the graph of possible worlds. We have provided a formal treatment of this approach by integrating a semantics of intervals into a standard modal temporal logics, and we have given an operational semantics for our system. Moreover, we have shown how the system can be implemented on top of a PROLOG system by defining a horizontal compilation scheme. We have demonstrated the usefulness of our system by giving an example which involves synchronization and coordination in a multi-agent scenario.

Outlook

From our current work quite a few new directives have arisen which will determine our future work in temporal logics. First, the combinatorial explosion caused by the operators which can be used to expand new worlds is a serious problem which appears in many areas in AI. It will force us to find and to apply intelligent methods of restricting the search space. The detection of cycles, the use of goal-driven search techniques, and the declaration of constraints by the user are possible steps in this direction. Second, due to our work in the field of multi-agent systems (cf. [BG88, GH89, DM90, DM91] for an overview), we will have to conduct further examinations on the role time plays with respect to multi-agent knowledge representation, distributed control and planning, as well as the coordination of the actions of different agents. The small example we presented in section 5 gives an idea of some of the possibilities contained in this field. After all, understanding the mystery of time continues to be one of the very fascinating challenges for human research.

Appendix A

The TRAM Solution for the Loading Dock Example

```
%=====
%
% A loading dock scenario
%
%=====
```

```
%*****
%***** Prim_ops ... *****
%*****
```

```
% An agent can DRIVE TO a LoadingPlatform, if
% it has to load something there
% it is driving
% and it did not come from this LoadingPlatform in the world before
```

```
prim_op( driveTo( Agent, LoadingPlatform ) ):-
  isAgent( Agent),
  isRamp( LoadingPlatform ),
  isDriving( Agent ),
  not done( driveTo( Agent, LoadingPlatform)),
  hasToLoadAtRamp( Agent, LoadingPlatform ).
```

```
%-----
```

```
% An agent can only be UNLOADED, if
% it stays at this LoadingPlatform
% this LoadingPlatform is its target LoadingPlatform
% it has nothing to load anywhere else.
```

```
prim_op( unload(Agent)) :-
```

```

isAgent( Agent),
isTargetRamp( Agent, LoadingPlatform ),
atRamp( Agent, LoadingPlatform ).

%-----

% It can only become LOADED, if
% it stays at this LoadingPlatform
% it still has to load something there

prim_op( load( Agent, LoadingPlatform )) :-
    isAgent( Agent),
    isRamp( LoadingPlatform ),
    atRamp( Agent, LoadingPlatform ),
    hasToLoadAtRamp( Agent, LoadingPlatform ).

%-----

% It can only MOVE TO THE LoadingPlatform, if
% it has entered the loading dock,
% there is no other agent staying at the LoadingPlatform,
% it has the order to load some goods at this LoadingPlatform
% it did not move back from the Platform the world before

prim_op( moveToRamp( Agent, LoadingPlatform)):-
    isAgent( Agent ),
    isRamp( LoadingPlatform),
    not done( moveBack( Agent, LoadingPlatform)),
    entered( Agent, LoadingPlatform ),
    not atRamp( _, LoadingPlatform),
    hasToLoadAtRamp( Agent, LoadingPlatform ).

%-----

% It can only MOVE AWAY FROM THE LoadingPlatform, if
% it stays there now

prim_op(moveBack( Agent, LoadingPlatform)):-
    isAgent( Agent ),
    isRamp( LoadingPlatform),
    not done(moveToRamp( Agent, LoadingPlatform)),
    atRamp( Agent, LoadingPlatform ).

%-----

% An agent has to CHANGE to TRamp, if
% it is already driving
% its order is to go by train from this TRamp

```

```

% its loadingOrder is empty

prim_op( changeToTrain( Agent, TRamp ) ):-
    isAgent( Agent),
    isTrainRamp( TRamp ),
    isDriving( Agent ),
    loadingOrder( Agent, []),
    hasToGoByTrain( Agent, Tramp, ToTRamp).

%-----

% An agent can only USE THE TRAIN, if
% it has to use it
% it sits in a train

prim_op( useTrain( Agent, FromTRamp, ToTRamp ) ):-
    isAgent( Agent),
    isTrainRamp( FromTRamp ),
    isTrainRamp( ToTRamp ),
    sitInTrain( Agent, FromTRamp ),
    hasToGoByTrain( Agent, FromTRamp, ToTRamp).

%-----

% It can only wait, if it stays in the loading dock; not at the ramp!!!

prim_op( wait( Agent ) ):-
    isAgent( Agent ), not atRamp( Agent, Goods), not done(wait(Agent)).

%*****
%***** Macro_ops ... *****
%*****

macro_op( getGoods( Agent, LoadingPlatform ) ) :-
    moveToRamp( Agent, LoadingPlatform),
    load( Agent, LoadingPlatform),
    moveBack( Agent, LoadingPlatform ).

macro_op( goByTrain( Agent, ToTRamp ) ) :-
    changeToTrain( Agent, FromTRamp),
    useTrain( Agent, FromTRamp, ToTRamp ).

%*****
%***** WORLD PREDICATES ... *****
%*****

```

```

:- world( startWorld ).
isDriving( truck1 ).
loadingOrder( truck1,[( LoadingPlatform1, g11 ),( LoadingPlatform2, g21 ) ] ).

:- world( startWorldTruck2 ).
isDriving( truck2 ).
loadingOrder( truck2,[( LoadingPlatform1, g12 ),( LoadingPlatform2, g22 ) ] ).

:- world( EveryWorld ).
isAgent( truck1 ).
hasToGoByTrain( truck1, rampT1, rampT2 ).
isTargetRamp( truck1, rampT2 ).

isAgent( truck2 ).
hasToGoByTrain( truck2, rampT1, rampT2 ).
isTargetRamp( truck2, rampT2 ).

isRamp( LoadingPlatform1 ).
isRamp( LoadingPlatform2 ).
isTrainRamp( rampT1 ).
isTrainRamp( rampT2 ).

hasToLoadAtRamp( Agent, LoadingPlatform ) :-
    isAgent( Agent ),
    isRamp( LoadingPlatform ),
    loadingOrder( Agent, List ),
    {member( (LoadingPlatform,Good), List )}. %%% This goal invokes original PROLOG

driveTo( Agent, LoadingPlatform ).entered( Agent, LoadingPlatform ).
driveTo( Agent, LoadingPlatform ).done( driveTo( Agent, LoadingPlatform ) ).
driveTo( Agent, LoadingPlatform ).loadingOrder( A, 0 ):-
    loadingOrder( A, 0 ).

moveToRamp( Agent, LoadingPlatform ).done( moveToRamp( Agent, LoadingPlatform ) ).
moveToRamp( Agent, LoadingPlatform ).atRamp( Agent, LoadingPlatform ).
moveToRamp( Agent, LoadingPlatform ).loadingOrder( A, 0 ):-
    loadingOrder( A, 0 ).

load( Agent, LoadingPlatform ).atRamp( Agent, LoadingPlatform ).
load( Agent, LoadingPlatform ).loadingOrder( Agent, NewOrder ):-
    loadingOrder( Agent, OldOrder ),
    {delete( OldOrder, (LoadingPlatform,Good), NewOrder)}. %%% ...

```

```

moveBack( Agent, LoadingPlatform).isDriving( Agent ).
moveBack( Agent, LoadingPlatform).done( moveBack( Agent, LoadingPlatform)).
moveBack( Agent, LoadingPlatform).loadingOrder( A, 0 ) :-
    loadingOrder( A, 0 ).

unload(Agent).orderDelete( Agent ).

changeToTrain( Agent, TRamp).sitInTrain( Agent, TRamp).
changeToTrain( Agent, TRamp).loadingOrder( A, 0 ):-
    loadingOrder( A, 0 ).

useTrain( Agent, FromTRamp, ToTRamp ).atRamp( Agent, ToTRamp).
useTrain( Agent, FromTRamp, ToTRamp ).loadingOrder( A, 0 ):-
    loadingOrder( A, 0 ).

goByTrain( Agent, ToTRamp ).atRamp( Agent, ToTRamp).
goByTrain( Agent, ToTRamp ).loadingOrder( A, 0 ):-
    loadingOrder( A, 0 ).

getGoods( Agent, LoadingPlatform).isDriving( Agent ).
getGoods( Agent, LoadingPlatform).loadingOrder( Agent, NewOrder ):-
    loadingOrder( Agent, OldOrder),
    {delete( OldOrder, (LoadingPlatform,Good), NewOrder)}. %%% ...

:- prolog.
thisWorld(W,W).
cconsult(loadingDock).
ccompile(loadingDock).

switch_to_world( startWorld ).
switch_to_world( startWorldTruck2 ).

apply( moveToRamp( Agent,X)).
apply( load( Agent, Ramp ) ).
apply( unload( Agent, Ramp ) ).
apply( moveBack( Agent,Y) ).
apply( driveTo( Agent, ramp1 )).
apply( driveTo( Agent, ramp2 )).
apply( changeToTrain( Agent, TRamp)).
apply( useTrain( Agent, FromRamp,ToRamp)).
apply( getGoods( Agent, Ramp,_ )).
apply( goByTrain( Agent, ToTRamp,_ )).

now( entered( Agent, Ramp ) ).
now( atRamp( Agent, Ramp ) ).

```

```

now( isDriving(Agent)).
now( loadingOrder( Agent, Order) ).
now( hasToGoByTrain( Agent, Start, Goal) ).
now( isTargetRamp( Agent, Ramp )).
now( hasToLoadAtRamp( Agent, Ramp )).
now( orderDelete(Agent) ).

```

```
f_diamond_trans( orderDelete(Agent), W , startWorld).
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compound goals %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%=====

```

```

Query One:
=====

```

```

Is there a World (Res1) in the future of (startWorld) in which
    "orderDelete( truck1 )" holds true and
is there a world (Res2) in the future of (startWorldTruck2) in which
    "orderDelete( truck2 )" holds true
and is there an interval (W1) in the past of the world (Res1), which has been
    created using "goByTrain(..)",
and is there an interval (W2) in the past of world (Res2), which has been
    created using "goByTrain(..)"
such that the intervals W1 and W2 are equal.

```

```

f_diamond_trans( orderDelete(truck1), Res1 , startWorld),
f_diamond_trans( orderDelete(truck2), Res2 , startWorldTruck2),
p_full_diamond_trans( thisWorld( W1 ) ,_, Res1),
    W1 = [goByTrain( truck1, rampT2, _ ) | _],
p_full_diamond_trans( thisWorld( W2 ) ,_, Res2),
    W2 = [goByTrain( truck2, rampT2, _ ) | _],
W1 equal W2.

```

```
%=====
```

```

Query Two
=====

```

```

% For no ramp (Ramp) there exist intervals (World_21) and (World_22) in the past
% of (Res1) and (Res2), which have been created by "getGoods" and between which
% the relations "" and "" hold. The predicates f_diamond_trans and p_full_diamond_trans
% correspond to the EML operators defined previously.

```

```
f_diamond_trans( orderDelete(truck1), Res1 , startWorld),
```

```

f_diamond_trans( orderDelete(truck2), Res2 , startWorldTruck2),
not( p_full_diamond_trans(
  (thisWorld( C1 ),
    { C1 =[getGoods(truck1,RampI,_)|_],
      p_full_diamond_trans( (thisWorld( C2 ),
        { C2=[getGoods(truck2,RampI,_)|_] } ),_,Res2),
      C1 equal C2
    }),_,Res1)
).

```

Result of the second query:

=====

```

** (2764) 0 Call:
getGoods(truck1,ramp1,
  [moveBack(truck1,ramp1),load(truck1,ramp1),moveToRamp(truck1,ramp1)])
.driveTo(truck1,ramp1)
.getGoods(truck1,ramp2,
  [moveBack(truck1,ramp2),load(truck1,ramp2),moveToRamp(truck1,ramp2)])
.driveTo(truck1,ramp2)
.startWorld
      equal
getGoods(truck2,ramp1,
  [moveBack(truck2,ramp1),load(truck2,ramp1),moveToRamp(truck2,ramp1)])
.driveTo(truck2,ramp1)
.getGoods(truck2,ramp2,
  [moveBack(truck2,ramp2),load(truck2,ramp2),moveToRamp(truck2,ramp2)])
.driveTo(truck2,ramp2)
.startWorldTruck2 ? 1
** (2940) 2 Call: relation(6,6,8,8,equal) ? 1
** (2940) 2 Exit: relation(6,6,8,8,equal) ?

```

===== First successful search, after which the world (Res2) is rejected.

```

** (3063) 0 Call:
getGoods(truck1,ramp1,
  [moveBack(truck1,ramp1),load(truck1,ramp1),moveToRamp(truck1,ramp1)])
.driveTo(truck1,ramp1)
.getGoods(truck1,ramp2,
  [moveBack(truck1,ramp2),load(truck1,ramp2),moveToRamp(truck1,ramp2)])
.driveTo(truck1,ramp2)
.startWorld
      equal
getGoods(truck2,ramp1,
  [moveBack(truck2,ramp1),load(truck2,ramp1),moveToRamp(truck2,ramp1)])
.driveTo(truck2,ramp1)
.startWorldTruck2 ? 1

```

```

** (3193) 2 Call: relation(6,2,8,4,equal) ? 1
** (3193) 2 Fail: relation(6,2,8,4,equal) ? 1

** (3285) 0 Call:
  getGoods(truck1,ramp2,
    [moveBack(truck1,ramp2),load(truck1,ramp2),moveToRamp(truck1,ramp2)])
  .driveTo(truck1,ramp2)
  .startWorld
    equal
  getGoods(truck2,ramp2,
    [moveBack(truck2,ramp2),load(truck2,ramp2),moveToRamp(truck2,ramp2)])
  .driveTo(truck2,ramp2)
  .getGoods(truck2,ramp1,
    [moveBack(truck2,ramp1),load(truck2,ramp1),moveToRamp(truck2,ramp1)])
  .driveTo(truck2,ramp1)
  .startWorldTruck2 ? 1
** (3415) 2 Call: relation(2,6,4,8,equal) ? 1
** (3415) 2 Fail: relation(2,6,4,8,equal) ? 1

```

=====
 ===== No contradiction could be found, so the resulting worlds
 ===== are printed out ...

```

Res1 =
  unload(truck1)
  .goByTrain(truck1,rampT2,
    [useTrain(truck1,rampT1,rampT2),changeToTrain(truck1,rampT1)])
  .getGoods(truck1,ramp1,
    [moveBack(truck1,ramp1),load(truck1,ramp1),moveToRamp(truck1,ramp1)])
  .driveTo(truck1,ramp1)
  .getGoods(truck1,ramp2,
    [moveBack(truck1,ramp2),load(truck1,ramp2),moveToRamp(truck1,ramp2)])
  .driveTo(truck1,ramp2)
  .startWorld,

```

```

Res2 =
  unload(truck2)
  .goByTrain(truck2,rampT2,
    [useTrain(truck2,rampT1,rampT2),changeToTrain(truck2,rampT1)])
  .getGoods(truck2,ramp2,
    [moveBack(truck2,ramp2),load(truck2,ramp2),moveToRamp(truck2,ramp2)])
  .driveTo(truck2,ramp2)
  .getGoods(truck2,ramp1,
    [moveBack(truck2,ramp1),load(truck2,ramp1),moveToRamp(truck2,ramp1)])
  .driveTo(truck2,ramp1)
  .startWorldTruck2,

```

C1 = _161,
RampI = _205,
C2 = _258

%%%

Bibliography

- [All83] J. F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11), 1983.
- [All84] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2), 1984.
- [AM89] M. Abadi and Z. Manna. Temporal logic programming. *Symbolic Computation*, (8):277–295, 1989.
- [BB87] H.J. Buerckert and K.H. Blaesius. *Deduktionssysteme*. Oldenbourg-Verlag, 1987.
- [BG88] A. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, Los Angeles, CA, 1988.
- [Brz89] C. Brzoska. Temporal logic programming A survey. *Institut fuer Logik, Komplexitaet und Deduktionssysteme, Universitaet Karlsruhe*, 1989.
- [CM81] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, 1981.
- [DM90] Y. Demazeau and J.-P. Müller. *Decentralized A.I.* North Holland, 1990.
- [DM91] Y. Demazeau and J.-P. Müller. *Decentralized A.I. 2*. North Holland, 1991.
- [Gab87] D. Gabbay. *Modal and temporal logic programming*. Academic Press, 1987.
- [Gal87] A. Galton. *Temporal logics and their applications*. Academic Press, 1987.
- [GH89] L. Gasser and M.N. Huhns. *Distributed Artificial Intelligence, Volume II*. Research Notes in Artificial Intelligence. Morgan Kaufmann, San Mateo, CA, 1989.
- [Hry88] T. Hrycej. Temporal PROLOG. In *Proc. of the 7th european conference on Artificial Intelligence*, pages 296–301, St. Paul, MN, 1988.
- [Kri71] S. Kripke. *Semantical considerations on modal logics*. Oxford University Press, 1971.
- [KS86] R.A. Kowalski and M.J. Sergot. A logic based calculus of events. *New generation computing*, (4):67–95, 1986.

- [Llo84] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.
- [MMPS92] H.J. Müller, J.P. Müller, M. Pischel, and R. Scheidhauer. On the Representation of Temporal Knowledge. Technical Report TR-92-??, DFKI Saarbrücken, 1992.
- [Nil80] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA, 1980.
- [OS89] H.J. Ohlbach and J.H. Siekmann. The Markgraf Karl Refutation Procedure. SEKI-report SR-89-19, Universität Kaiserslautern, 1989.
- [Pis91] M. Pischel. TRAM: Integration verschiedener Ansätze zur Zeitrepräsentation. Master's thesis, Universität Kaiserslautern, 1991.
- [Pri67] A.N. Prior. *Past, presence and future*. Clarendon Press, Oxford, 1967.
- [Ram88] A. Ramsay. *Formal methods in Artificial Intelligence*. Cambridge University Press, 1988.
- [RK91] E. Rich and K. Knight. *Artificial Intelligence*. McGraw Hill, 2nd edition, 1991.
- [Sch89] R. Scheidhauer. Chronolog, 1989.
- [Sho88] Y. Shoham. *Reasoning about change: Time and causation from the standpoint of AI*. MIT Press, 1988.
- [SM87] Y. Shoham and D.V. McDermott. Temporal reasoning, 1987.
- [SS86] L. Sterling and E. Shapiro. *The Art of Prolog*. MIT Press, Cambridge, Massachusetts, 1986.
- [Tan89] T.G. Tang. Temporal logic CTL + PROLOG. *Journal of Automated Reasoning*, (5):49–65, 1989.



DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.

Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.

The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-91-10

Franz Baader, Philipp Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages
31 pages

RR-91-11

Bernhard Nebel: Belief Revision and Default Reasoning: Syntax-Based Approaches
37 pages

RR-91-12

J.Mark Gawron, John Nerbonne, Stanley Peters: The Absorption Principle and E-Type Anaphora
33 pages

RR-91-13

Gert Smolka: Residuation and Guarded Rules for Constraint Logic Programming
17 pages

RR-91-14

Peter Breuer, Jürgen Müller: A Two Level Representation for Spatial Relations, Part I
27 pages

RR-91-15

Bernhard Nebel, Gert Smolka: Attributive Description Formalisms ... and the Rest of the World
20 pages

RR-91-16

Stephan Busemann: Using Pattern-Action Rules for the Generation of GPSG Structures from Separate Semantic Representations
18 pages

RR-91-17

Andreas Dengel, Nelson M. Mattos: The Use of Abstraction Concepts for Representing and Structuring Documents
17 pages

RR-91-18

John Nerbonne, Klaus Netter, Abdel Kader Diagne, Ludwig Dickmann, Judith Klein: A Diagnostic Tool for German Syntax
20 pages

RR-91-19

Munindar P. Singh: On the Commitments and Precommitments of Limited Agents
15 pages

RR-91-20

Christoph Klauck, Ansgar Bernardi, Ralf Legleitner: FEAT-Rep: Representing Features in CAD/CAM
48 pages

RR-91-21

Klaus Netter: Clause Union and Verb Raising Phenomena in German
38 pages

RR-91-22

Andreas Dengel: Self-Adapting Structuring and Representation of Space
27 pages

RR-91-23

Michael Richter, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Akquisition und Repräsentation von technischem Wissen für Planungsaufgaben im Bereich der Fertigungstechnik
24 Seiten

RR-91-24

Jochen Heinsohn: A Hybrid Approach for Modeling Uncertainty in Terminological Logics
22 pages

RR-91-25

Karin Harbusch, Wolfgang Finkler, Anne Schauder:
Incremental Syntax Generation with Tree Adjoining
Grammars
16 pages

RR-91-26

*M. Bauer, S. Biundo, D. Dengler, M. Hecking,
J. Koehler, G. Merziger:*
Integrated Plan Generation and Recognition
- A Logic-Based Approach -
17 pages

RR-91-27

*A. Bernardi, H. Boley, Ph. Hanschke,
K. Hinkelmann, Ch. Klauck, O. Kühn,
R. Legleitner, M. Meyer, M. M. Richter,
F. Schmalhofer, G. Schmidt, W. Sommer:*
ARC-TEC: Acquisition, Representation and
Compilation of Technical Knowledge
18 pages

RR-91-28

Rolf Backofen, Harald Trost, Hans Uszkoreit:
Linking Typed Feature Formalisms and
Terminological Knowledge Representation
Languages in Natural Language Front-Ends
11 pages

RR-91-29

Hans Uszkoreit: Strategies for Adding Control
Information to Declarative Grammars
17 pages

RR-91-30

Dan Flickinger, John Nerbonne:
Inheritance and Complementation: A Case Study of
Easy Adjectives and Related Nouns
39 pages

RR-91-31

H.-U. Krieger, J. Nerbonne:
Feature-Based Inheritance Networks for
Computational Lexicons
11 pages

RR-91-32

Rolf Backofen, Lutz Euler, Günther Görz:
Towards the Integration of Functions, Relations and
Types in an AI Programming Language
14 pages

RR-91-33

Franz Baader, Klaus Schulz:
Unification in the Union of Disjoint Equational
Theories: Combining Decision Procedures
33 pages

RR-91-34

Bernhard Nebel, Christer Bäckström:
On the Computational Complexity of Temporal
Projection and some related Problems
35 pages

RR-91-35

Winfried Graf, Wolfgang Maaß: Constraint-basierte
Verarbeitung graphischen Wissens
14 Seiten

RR-92-01

Werner Nutt: Unification in Monoidal Theories is
Solving Linear Equations over Semirings
57 pages

RR-92-02

*Andreas Dengel, Rainer Bleisinger, Rainer Hoch,
Frank Hönes, Frank Fein, Michael Malburg:*
 Π_{ODA} : The Paper Interface to ODA
53 pages

RR-92-03

Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04

John Nerbonne: Feature-Based Lexicons:
An Example and a Comparison to DATR
15 pages

RR-92-05

*Ansgar Bernardi, Christoph Klauck,
Ralf Legleitner, Michael Schulte, Rainer Stark:*
Feature based Integration of CAD and CAPP
19 pages

RR-92-06

Achim Schupetea: Main Topics od DAI: A Review
38 pages

RR-92-07

Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08

Gabriele Merziger: Approaches to Abductive
Reasoning - An Overview -
46 pages

RR-92-09

Winfried Graf, Markus A. Thies:
Perspektiven zur Kombination von automatischem
Animationsdesign und planbasierter Hilfe
15 Seiten

RR-92-11

Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command
Language Environment
13 pages

RR-92-13

Markus A. Thies, Frank Berger:
Planbasierte graphische Hilfe in objektorientierten
Benutzungsoberflächen
13 Seiten

RR-92-14

Intelligent User Support in Graphical User Interfaces:

1. InCome: A System to Navigate through Interactions and Plans
Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
Markus A. Thies, Frank Berger

22 pages

RR-92-15

Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations

23 pages

RR-92-16

Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, Hans-Jürgen Profüllich: An Empirical Analysis of Terminological Representation Systems

38 pages

RR-92-17

Hassan Ait-Kaci, Andreas Podelski, Gert Smolka: A Feature-based Constraint System for Logic Programming with Entailment

23 pages

RR-92-18

John Nerbonne: Constraint-Based Semantics

21 pages

RR-92-19

Ralf Legleitner, Ansgar Bernardi, Christoph Klauck: PIM: Planning In Manufacturing using Skeletal Plans and Features

17 pages

RR-92-20

John Nerbonne: Representing Grammar, Meaning and Knowledge

18 pages

RR-92-21

Jörg-Peter Mohren, Jürgen Müller: Representing Spatial Relations (Part II) -The Geometrical Approach

25 pages

RR-92-22

Jörg Würtz: Unifying Cycles

24 pages

RR-92-24

Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain

20 pages

DFKI Technical Memos**TM-91-09**

Munindar P. Singh: On the Semantics of Protocols Among Distributed Intelligent Agents
18 pages

TM-91-10

Béla Buschauer, Peter Poller, Anne Schauder, Karin Harbusch: Tree Adjoining Grammars mit Unifikation
149 pages

TM-91-11

Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

TM-91-12

Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13

Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14

Rainer Bleisinger, Rainer Hoch, Andreas Dengel: ODA-based modeling for document analysis
14 pages

TM-91-15

Stefan Bussmann: Prototypical Concept Formation An Alternative Approach to Knowledge Representation
28 pages

TM-92-01

Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02

Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03

Mona Singh: A Cognitive Analysis of Event Structure
21 pages

TM-92-04

Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer: On the Representation of Temporal Knowledge
61 pages

DFKI Documents**D-91-10**

Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd
European Workshop on „Modeling Autonomous
Agents and Multi-Agent Worlds“
246 pages

Note: This document is available only for a
nominal charge of 25 DM (or 15 US-\$).

D-91-11

Thilo C. Horstmann: Distributed Truth Maintenance
61 pages

D-91-12

Bernd Bachmann:
HieraCon - a Knowledge Representation System
with Typed Hierarchies and Constraints
75 pages

D-91-13

International Workshop on Terminological Logics
*Organizers: Bernhard Nebel, Christof Peltason,
Kai von Luck*
131 pages

D-91-14

*Erich Achilles, Bernhard Hollunder, Armin Laux,
Jörg-Peter Mohren: KRIS: Knowledge
Representation and Inference System*
- Benutzerhandbuch -
28 Seiten

D-91-15

*Harold Boley, Philipp Hanschke, Martin Harm,
Knut Hinkelmann, Thomas Labisch, Manfred
Meyer, Jörg Müller, Thomas Oltzen, Michael
Sintek, Werner Stein, Frank Steinle:*
 μ CAD2NC: A Declarative Lathe-Worplanning
Model Transforming CAD-like Geometries into
Abstract NC Programs
100 pages

D-91-16

Jörg Thoben, Franz Schmalhofer, Thomas Reinartz:
Wiederholungs-, Varianten- und Neuplanung bei der
Fertigung rotationssymmetrischer Drehteile
134 Seiten

D-91-17

Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick
auf ihre Eignung für die Arbeitsplanung
86 Seiten

D-91-18

Thomas Reinartz: Definition von Problemklassen
im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19

Peter Wazinski: Objektlokalisierung in graphischen
Darstellungen
110 Seiten

D-92-01

Stefan Bussmann: Simulation Environment for
Multi-Agent Worlds - Benutzeranleitung
50 Seiten

D-92-02

Wolfgang Maaß: Constraint-basierte Platzierung in
multimodalen Dokumenten am Beispiel des Layout-
Managers in WIP
111 Seiten

D-92-03

*Wolfgang Maaß, Thomas Schiffmann, Dudung
Soetopo, Winfried Graf:* LAYLAB: Ein System zur
automatischen Platzierung von Text-Bild-
Kombinationen in multimodalen Dokumenten
41 Seiten

D-92-06

Hans Werner Höper: Systematik zur Beschreibung
von Werkstücken in der Terminologie der
Featuresprache
392 Seiten

D-92-08

Jochen Heinsohn, Bernhard Hollunder (Eds.):
DFKI Workshop on Taxonomic Reasoning
Proceedings
56 pages

D-92-09

Gernod P. Laufkötter: Implementierungsmöglich-
keiten der integrativen Wissensakquisitionsmethode
des ARC-TEC-Projektes
86 Seiten

D-92-10

Jakob Mauss: Ein heuristisch gesteuerter Chrat-
Parser für attributierte Graph-Grammatiken
87 Seiten

D-92-15

DFKI Wissenschaftlich-Technischer Jahresbericht
1991
130 Seiten

D-92-21

Anne Schauder: Incremental Syntactic Generation of
Natural Language with Tree Adjoining Grammars
57 pages

