Abstract

The area of reasoning about action and change is concerned with the formalization of actions and their effects as well as other aspects of inhabited dynamical systems. The representation is typically done in some logical language. Although there has been substantial progress recently regarding the frame problem and the ramification problem, many problems still remain. One of these problems is the representation of concurrent actions and their effects. In particular, the effects of two or more actions executed concurrently may be different from the union of the effects of the individual actions had they been executed in isolation. This thesis presents a language, TAL-C, which supports detailed and flexible yet modular descriptions of concurrent interactions. Two related topics, which both require a solution to the concurrency problem, are also addressed: the representation of effects of actions that occur with some delay, and the representation of actions that are caused by other actions.

Another aspect of reasoning about action and change is how to describe higher-level reasoning tasks such as planning and explanation. In such cases, it is important not to just be able to reason about a specific narrative (course of action), but to reason about alternative narratives and their properties, to compare and manipulate narratives, and to reason about alternative results of a specific narrative. This subject is addressed in the context of the situation calculus, where it is shown how the standard version provides insufficient support for reasoning about alternative results, and an alternative version is proposed. The narrative logic NL is also presented; it is based on the temporal action logic TAL, where narratives are represented as first-order terms. NL supports reasoning about (I) metric time, (II) alternative ways the world can develop relative to a specific choice of actions, and (III) alternative choices of actions.

Acknowledgments

First, I would like to thank my supervisor, Patrick Doherty, for his support. I would also like to thank the other two persons in my advisory group: Erik Sandewall and Ulf Nilsson.

I have greatly enjoyed working with Joakim Gustafsson, with whom I have co-authored two of the papers in this thesis. Patrick Doherty, who also co-authored one of these papers, should be mentioned here as well. Many other people have also contributed with comments to and discussion on the work presented here: Marcus Bjäreland, Patrik Haslum, Andrzej Szałas, Jonas Kvarnström, Silvia Coradeschi, and a number of anonymous reviewers.

I want to express my gratitude also to those persons who have helped me with various practical issues, and among those in particular Lillemor Wallgren, Lise-Lott Svensson, Kit Burmeister, Janette de Castro, Anna-Maria Uhlin, and the people at TUS. And, of course, Ivan Rankin, who has debugged my English (and if there are any faults left, then I'm the one to blame.) And while I'm at it, I should also take the opportunity to thank all the people at IDA for providing such a pleasant environment.

Finally, I would like to thank Silvia Coradeschi once more, this time in the capacity as my life companion, for all her support and patience during the time I have worked on this thesis.

This work has been supported by the Wallenberg foundation and by the Swedish Research Council for Engineering Sciences (TFR).

Contents

Part I
Background

Chapter 1

Introduction

The field of reasoning about action and change (RAC for short) is concerned with the modeling of dynamical systems. A model is a tool with which one can answer questions about a system without actually having to manipulate and experiment with the system. In RAC, the types of models that one is interested in are typically intended to be used to reason about actions executed by agents and how these actions affect the system or environment in which they are performed. In this thesis, we address two aspects of RAC. The first aspect is trying to determine the outcome of some given course of action; here, we investigate how to reason about actions and processes that occur concurrently with other actions or processes. The second aspect concerns reasoning about alternative courses of action and their consequences; this involves such issues as combining different courses of actions.

There are a number of potential applications for RAC, such as fault diagnosis for complicated machinery and natural language understanding. The most important purpose of RAC, as envisioned by McCarthy and Hayes as early as in their 1969 paper [97] is to provide representations that can serve as a basis for reasoning and acting for artificial autonomous agents. These agents include both robots that operate in physical environments and softbots that operate entirely inside information systems. An agent is equipped with sensors which can provide it with information about its environment. The sensor types might be video cameras, sonars, or, in the case of a softbot, operations for accessing the contents of a file. The agent also has actuators that allow it to manipulate its environment. These actuator types might be grippers, wheels, or in the case of a softbot, operations for modifying a file.[1]

[1]Notice that much knowledge can be built into the sensors and actuators, for instance how to move the gripper in order to pick up a can, and the visual signature (color codes,

Further, an agent typically has some limited purpose, such as some class of goals or tasks that it is intended to pursue.

By equipping an agent with the capability to represent and reason about the dynamics of the world, it can predict the consequences of its actions before it actually executes them. In the words of Davis, Shrobe and Szolovitz [23], the representation serves as a substitute for reality. Thereby, the agent can pick actions that are purposeful and avoid actions that are harmful in a given situation. It need not be limited to just its own next action, but can reason about actions further ahead in time, and also about actions of other agents and natural processes. The agent may also reason backwards in time, seeking explanations for how certain conditions came about, in particular when something has gone wrong. Finally, the agent may reason about facts that are not immediately accessible to its sensors at a given moment; it can complement what it can observe of the present with its knowledge about the past, its general background knowledge and defeasible reasoning.

1.1 Logics of action and change

Recall that a model of a system is a tool which can be used to answer questions about the system. If we intend to model a dynamical environment for an agent, a number of things have to be taken into consideration.

- In order to be implementable, a model must be precisely and rigorously defined.
- As it might be used for different modes of reasoning, it is advantageous if a model is constructed in a way that does not impose unnecessary constraints on how it can be applied. That is to say, one wants to separate facts from reasoning processes. Therefore, it is preferable that the model is declarative and is supplied with some kind of general inference mechanism. A less reasonable alternative is to make the model procedural, in which case the choices of what inferences to make are hard-wired into it.
- The model should include aspects of the environment (and the agents themselves) that are relevant to the actions the agent can perform and that are useful for pursuing the goals and executing the tasks of the agent.

geometry) of that can. This type of knowledge is often not appropriate for high-level reasoning and decision making, though.

In reasoning about action and change, as in many other branches of AI, the preferred modeling language is typically some logic, such as first-order predicate logic or some modal logic. Logics are indeed rigorous formal constructions, they are declarative, and they are largely independent of the subject matter, and can be used for defining arbitrary concepts. In addition, logics support compact representation of both incomplete and general information.

A not uncommon objection against RAC is that logics are not necessary; one can simply use equations from physics for describing the dynamics of the world. However, a purely physical model does not always satisfy the third point above. It would contain much information that would not be appropriate for decision making but that would rather be built into the actuators and sensors of the agent. It would also lack many concepts that are releavant for the agent, or make the description of such concepts unnecessarily complicated. For instance, consider a robot delivering mail. A purely physical description of what a letter is would leave out some very important characteristics of letters, such as that it has an addressee. The physics of picking up and putting down letters is rarely anything the agent needs to reason about in any detail.

Although logics are highly flexible tools, how to actually use them for encoding knowledge about dynamical systems is an issue that has been far from being resolved. It is an issue that both involves a number of difficult technical problems — the most well-known of them are the frame, ramification and qualification problems — and a number of difficult questions of a more conceptual nature, such as what temporal structures to use and what properties actions should have. There has been some significant progress in RAC. For instance, there exist a number of concise and fairly general solutions for the frame problem. But there are still many problems left to solve before RAC can be applied to many realistic domains, in particular those domains where robots operate in complex physical environments.

1.2 Aim of this thesis

The aim of this thesis is to extend the capabilities of a particular approach to RAC. There are essentially two problems that are addressed, namely how to represent such phenomena as action concurrency and delayed effects, and how to represent what happens in the world in a way that permits us to reason about alternative courses of actions.

1.2.1 Concurrent interactions, delayed effects, and causality between actions

The first problem is how to represent a number of important interrelated phenomena which have not previously been satisfactorily addressed, namely concurrent interactions between actions and causal dependencies, delayed effect of actions, and action occurrences that are caused by other action occurrences. These subjects, all fundamental to the representation of realistic dynamical systems, are studied in the context of the Temporal Action Logic (TAL for short) [124, 26]. Although the choice of TAL was a natural consequence of the author's affiliation, it also has some more objective rationale. In particular, TAL has metric time and actions have temporal duration, which is a significant advantage when it comes to dealing with true concurrency and delays. As a matter of fact, we would claim that a meaningful treatment of concurrent interactions between actions requires that actions are considered to have temporal extension. Likewise, a meaningful treatment of delayed effects and triggered action occurrences requires metric time and representation of concurrent interactions. The fact that one or several of these properties are lacking or have only recently been added to many popular RAC languages perhaps explains the lack of progress regarding these subjects.

1.2.2 Reasoning about narratives

The second problem is the representation of action sequences and narratives as first-order objects. A narrative in a language like TAL is a set of sentences that describe what actions actually occur in the world, and the temporal properties of and relations between these actions. In addition, a narrative can also contain observations relating to the actual state of the world at different points in time. An objection against narrative-based languages like TAL is that they do not permit reasoning about higher-order action structures (e.g. action sequences and narratives) as first-order objects. It has been claimed by for instance Reiter [120] that this makes narrative-based languages such as TAL less suitable for reasoning tasks such as planning, where one needs to reason about alternative narratives. Planning has to be done abductively in TAL, as opposed to languages with a branching event-based time such as situation calculus [97, 118], in which action sequences are first-order objects and planning can be done purely deductively. We present results that show that the ability of situation calculus to represent plans is more restricted than is generally acknowledged. In particular if the initial

situation[2] is only partially specified or actions can have nondeterministic effects,[3] then traditional situation calculus versions [118] are insufficient. We provide a modified version of situation calculus that manages to solve these problems.

The branching event-based temporal structure of situation calculus has some drawbacks; in particular, there is no notion of time which is independent of actions. There have been a number of attempts to integrate a metric time-line and other narrative-style features into situation calculus, but with moderate success. Instead, we present an attempt to start from the opposite direction and integrate situation calculus-style features (action sequences/narratives as first-order objects) into a narrative-based language. The result is the narrative logic (NL for short), which is an extension of TAL that permits reasoning about narratives as first-order objects.

1.3 Organization of the thesis

The thesis consists of two parts. The first part, chapters 1 to 5, is an introduction to reasoning about action and change, and an in-depth analysis of a number of approaches to the problems addressed in this thesis. The second part consists of a collection of (with one exception) published articles — papers I to V — that contain the main scientific contribution of this thesis. The versions of these papers presented here are the same as those originally published, apart from some minor rephrasings.

Chapter 1. Introduction. We present an informal introduction to reasoning about action and change and the problems addressed in this thesis.

Chapter 2. Problems in reasoning about action and change. We provide an introduction to reasoning about action and change and the types of problems addressed in the area. In particular, we discuss different ontological and epistemological choices that can be made when designing a logic of action and change. We also address four difficult technical problems of action and change, namely the frame problem, the ramification problem, the qualification problem and the concurrency problem.

[2] The initial situation in a situation calculus axiomatization determines what facts hold before any actions have been executed.

[3] An action has nondeterministic effects if one cannot determine what the result will be even if one has perfect information about the state before the action. One example is rolling a die.

Chapter 3. Languages for reasoning about action and change. The basic versions of a number of different representations are presented and analyzed: the situation calculus [97, 118] and its relatives $\mathcal{A}$ [43], the fluent calculus [55, 18] and $\mathcal{L}_1$ [13], the event calculus [69, 129], the temporal action logic [124, 26, 29], and modal logics such as dynamic logic [116] and the computation tree logic CTL* [21].

Chapter 4. Concurrent interactions. This chapter presents and analyzes a number of approaches to concurrent actions and interactions by Georgeff [45], Pinto [112, 113], Baral and Gelfond [11], Bornscheuer and Thielscher [18], and Pelavin [109]. The purpose of this chapter is to complement paper I, which is about concurrent interactions, and to provide more detailed presentations of a number of approaches to concurrency.

Chapter 5. Enriching the representation of narratives. There have been a number of attempts to combine the expressive powers of languages with branching event-driven time and narrative based languages with explicit time. In the first half of this chapter, we investigate three such approaches: two by Pinto [112, 113] and one by Reiter [120]. All three are approaches that extend situation calculus with explicit time. This investigation is intended to provide a background to paper V. The latter two of these proposals also includes the possibility to have triggered actions and events.

Paper I. Reasoning about concurrent interaction. We present an approach for representing concurrent interactions in TAL. The approach involves only minor modifications of TAL, and the extended language is called TAL-C. In particular, a new type of feature, called durational features, is introduced. It makes extensive use of dependency laws, and uses influence features as intermediates for describing concurrent interactions. We show how TAL-C can represent a range of different kinds of concurrent interactions, and we also argue that the approach has good modularity properties.

This paper is a result of joint work with Joakim Gustafsson and was published in the Journal of Logic and Computation, 1999.

Paper II. Delayed Effects of Actions. A fundamental property of many dynamical systems is that effects of actions can occur with some delay. In this paper, we address the representation of delayed effects in the context of reasoning about action and change. We discuss how delayed

effects can be modeled both in abstract ways and as detailed processes, and we consider a range of possible interactions between the delayed effect of an action and later occurring actions, including interference and cumulative effects.

This paper is a result of joint work with Joakim Gustafsson and Patrick Doherty, and was presented at ECAI'98.

Paper III. Causal relationships between actions in TAL. This is an unpublished manuscript that addresses more complex types of occurrences, such as occurrences triggered by specific conditions or by other occurrences, or that are prevented by other occurrences. The paper can be viewed as a complement to paper II — sometimes it might be more appropriate to view a response to an action as a triggered event occurrence rather than as a delayed effect.

Paper IV. Reasoning about incomplete information and non-determinism in situation calculus. One of the main advantages of (Reiter's [118]) situation calculus is that action sequences are first-order terms. In particular, planning can be formulated as an existence problem. This paper shows how these properties break down when incomplete information about the initial state and nondeterministic action effects are introduced, basically due to the fact that this incompleteness is not adequately manifested on the object level. A version of situation calculus is presented which adequately models the alternative ways the world can develop relative to a choice of actions.

This paper was presented at IJCAI'97.

Paper V. Anything can happen: on narratives and hypothetical reasoning. This paper presents an extension of the temporal action logic (TAL) where narratives are represented as terms. This permits us to combine into one single formalism the capabilities to represent and reason about (I) metric time, (II) alternative ways the world can develop relative to a specific choice of actions, and (III) alternative choices of actions.

This paper was presented at KR'98.

Chapter 2

Problems in reasoning about action and change

An agent that is to operate in a dynamic world in something more than a merely reactive manner can benefit from being able to reason about the way the world develops over time. If this reasoning is performed symbolically, the agent needs some kind of symbolic representation of the world and how it changes. During the design of such a representation, a number of choices have to be made. These choices are relevant both for the classes of worlds that should be represented and for the types of reasoning that are to be done. In this chapter, we explore some of the choices that have to be made during the design of an action-and-change representation. We also discuss a number of technical problems one can encounter during this design, namely the frame, ramification, qualification and concurrency problems.

2.1 Ontological considerations

The process of designing a representation always involves an element of abstraction; some aspects of the world are considered relevant and are introduced as concepts in the representation, and others are not. The choice of concepts and how they are structured will be called the ontology of the representation. Factors that determine the choice of ontology include the actual structure of the represented environment and the purpose of the representation. Regarding the structure, we distinguish between the elementary concepts that are the basic building blocks in our ontology, and the structural concepts that allow us to relate and combine these basic building blocks in different ways.

2.1.1 Elementary concepts

Typical elementary concepts in reasoning about action and change are as follows.

1. Objects (individuals).

2. Agents, a special class of objects that are capable of performing actions in the world.

3. Features, or state variables, or fluents, representing time-variant properties and relations on the objects in the world. Features are usually associated with value domains, which can be, for instance, boolean values.

4. Actions, events, and other kinds of occurrences that cause fluents to change.

5. Time points, or time intervals.

2.1.2 Structural concepts

These elementary concepts can then have properties and be related in different ways. One important issue is the relation between time and action occurrences. Time can either be something independent, something "in itself", or it can be generated by the actions and events of the world. In the former case, a metric linear structure such as the integers or the reals is a possible choice [124, 69]. Another possibility is a qualitative (or mixed qualitative-quantitative) temporal structure based on intervals [5]. An independent notion of time facilitates the representation of non-trivial temporal relations between action instances and other time-dependent facts about the world. In the latter case, the temporal entities are denoted by sequences of actions or events [97], or more complex structures. This results in a branching temporal structure, where each branch represents one possible course of actions.

The value assignments to a given set of features at a point in time is commonly referred to as a state. A sequence of states over time is sometimes called a development. The way the state changes over time is affected by the occurrence of actions and events. Actions often influence states in some regular ways; in particular, they affect only a limited number of features and for a limited period of time. Other common types of regularities include

nondeterministic vs. deterministic effects, context-dependent vs. context-independent effects and indirect effects (ramifications). Furthermore, if actions can occur concurrently, then actions might affect other actions and effects might interact. For instance, one action can prevent the execution of another action.

Sandewall has proposed an ontological taxonomy [124], which mainly concerns the properties of actions and the way they affect the world. The following are some useful properties from that taxonomy.

Strong inertia. Features only change due to (direct or indirect) effects of actions. Features do not spontaneously change values.

Alternative results of actions. Actions do not always produce the same effects. This can be due to nondeterminism (see below) or context-dependency (executing the action in different states can result in different effects.) Example: The effects of dropping an object depend on how fragile the object is and where it is dropped.

Deterministic effects. Whenever the action is executed in a state, it always has the same effects. If an action does not have this property, it is said to have nondeterministic effects. Nondeterministic effects can both be of varying duration (see below) and assign varying values to features. Example: throwing a die is typically modeled as a nondeterministic action.

Extended actions. Actions take time to execute, and their effects occur over time. Example: walking from one room to another usually takes time. The opposite case is single-step actions, whose effects are almost immediate or instantaneous, depending on the temporal structure used.

Encapsulated actions. Although actions might take time, they are only characterized in terms of the preconditions and effects; what happens inside the duration of an action is irrelevant. For non-encapsulated actions, on the other hand, what happens while the action is executed is relevant.

Ramification. Features not only change as the result of the direct effects of actions, but also change due to dependencies with other features. Example: Turning the ignition key of a car starts the electrical system, which in turn starts the radio, the coupe light and the air conditioning.

Concurrency. Actions can occur simultaneously, and the duration of actions can overlap.

Delayed effects. Changes due to an action may occur at a later time.

Surprises. Changes that are not due to actions can occur, although infrequently.

Normality. Certain (partial) states occur more frequently than others.

Paper I in this thesis addresses concurrency, and paper II addresses delayed effects. There are also some additional ontological properties that will be addressed in this thesis: alternative behaviors of features, in particular behaviors with default values vs. persistence (also in paper I) and causal dependencies between actions/events such as when an event is triggered by some action (chapter 5.)

2.2 Epistemological considerations

A representation might leave more or less room for uncertainty and even inaccuracy. For instance, the representation might require that there is information about the value assignment for all features at the initial time-point (provided there is one), or it might require that all effects of an action are completely and correctly specified. Further, uncertainty might be graded, for instance in terms of probabilities. Sandewall has also proposed a taxonomy [124] of epistemological specialties. These specialties refer to the relation between information in a scenario description and the actual world. A scenario description is a set of statements such as a description of the effects of actions, action occurrences, and observations about the state of the world at different points in time. The following is a sample of epistemological specialties:

Accurate knowledge. The effects of actions are completely and correctly described, all action occurrences are described (although their timing might be unspecified), and all observations are correct.

Accurate knowledge with complete initial state. As above, but in addition the initial state is completely specified.

Qualified action descriptions. The effects of an action may be unspecified when certain preconditions of the action do not hold.

Uninformed about actions. Action occurrences may be missing.

Uninformed about action laws. The effects of actions are incompletely described.

Misconceptions in observations. Observations might be incorrect.

This thesis does not address these epistemological issues in any depth.

2.3 Higher-level reasoning

The issues we have addressed so far relate to how a representation reflects the structure of the environment and of particular courses of actions and the quality of the information it encodes. Another important aspect is how the representation can be used to represent higher-level reasoning tasks such as planning and explanation. For instance, in order to do planning, one needs to reason about alternative courses of action and to evaluate their outcome, one might have to analyze flaws in the plan and decide how to fix them, and one might have to combine different subplans. We do not attempt to make an exhaustive enumeration, but the following are some interesting aspects of higher-level reasoning that are relevant to this thesis (in particular to papers IV and V).

Reasoning about alternative courses of action. For instance, how one can formulate "the execution of these actions leads to the result α, but the execution of these other actions leads to the result $\neg\alpha$".

Combining (merging) several courses of action. For instance, how one can formulate "if the actions in A lead to α, and the actions in A' lead to β given that α holds in the beginning, then A and A' together, in sequence, lead to β".

Reasoning about alternative developments of the same course of action. For instance, how one can formulate "the execution of these actions always/necessarily leads to the result α", or "the execution of these actions sometimes/possibly leads to the result $\neg\alpha$."

2.4 Problems in representing change

Assuming that one has fixed the ontological and epistemological assumptions under which the representation is intended to be used, it still remains to actually pin down the representation itself and how it captures the dynamics and structures of what it is intended to represent. At the procedural

extreme, this can be hardwired in a simulator, which computes things in a very fixed manner. At the declarative extreme, it can be axiomatized in some suitable logic. Finding such an axiomatization has been the topic for most of the research in the field of reasoning about action and change. Historically, three problems of such an axiomatization have been considered, namely the frame, ramification and qualification problems. We will also discuss a fourth problem, which we call the concurrency problem.

2.5 The frame problem

The frame problem was first described by McCarthy and Hayes [97] in the context of the situation calculus. It concerns how to avoid the need to explicitly specify all the aspects of the world that do not change when an action is performed. Consider the following example about buying coffee from a coffee vending machine.

> *Initially, the agent has no coffee and no credits in the machine.*
> *If the agent draws its coffee card through the card reader of the machine, a credit is registered in the next situation.*
> *If a credit is registered and the button is pressed, then the credit is lost and the agent gets a cup of coffee.*
> *There is also a waiting action, which does not have any effects.*

Assume that first the card is drawn, then the agent waits and then the button is pressed. Can one now, based solely on the information above, arrive at the conclusion that the result is that the agent has got a cup of coffee? Unfortunately not, at least if one is restricted to using only explicitly stated information. The reason is that there is no explicit information stating that there still is a credit after the waiting action is performed, and therefore one cannot conclude that the coffee will be there after pressing the button. It would be reasonable to assume that the credit still remained after waiting, but the point is that this has not been explicitly stated. The encoding of this kind of tacit, commonsense information is one of the central issues in AI. In this particular case, the remedy is to add the following rule.

> *If an aspect of the world is not explicitly changed by an action, then one can assume that it remains the same after the action is invoked and executed.*

Using this rule of thumb, one can arrive at the conclusion that there still is credit left after waiting, and that the agent will get its coffee when the

button is subsequently pressed. Actually, one can argue that the assumption that things remain the same is a gross oversimplification. Indeed, many properties and relations change without being affected by actions: wet things dry, hot things cool off and so on. Thus, it would be more realistic to state something like "*If aspect X of the world is not explicitly affected by any action, then one can assume that it behaves as follows...*". However, descriptions of the frame problem have always assumed that the default behavior of properties and relations is no-change, and the remainder of this chapter will also be based on this assumption.

2.5.1 Frame axioms

Now, assume that the same facts are expressed in a logic-based formalism, such as situation calculus [97, 118]. The first three statements can be translated as follows (the fourth one about the waiting action does not have to be axiomatized).

$$\neg Holds(\textit{has-coffee}, S0) \wedge \neg Holds(\textit{credit}, S0) \tag{2.1}$$

$$\forall s[Holds(\textit{credit}, result(\textit{DrawCard}, s))] \tag{2.2}$$

$$\begin{aligned}&\forall s[Holds(\textit{credit}, s) \Rightarrow \\ &\quad (Holds(\textit{has-coffee}, result(\textit{PressButton}, s)) \wedge \\ &\quad \neg Holds(\textit{credit}, result(\textit{PressButton}, s)))]\end{aligned} \tag{2.3}$$

Sentence (2.1) is an observation about the initial situation, denoted by the symbol $S0$. Sentences (2.2–2.3) are effect axioms for the actions *DrawCard* and *PressButton*, where $result(\textit{DrawCard}, s)$ and $result(\textit{PressButton}, s)$ are terms denoting the situations after drawing the card and pressing the button respectively in situation s. The predicate $Holds$ represents what facts hold in a given situation. The statement $Holds(\textit{credit}, result(\textit{DrawCard}, s))$, for instance, denotes that the property (fluent) *credit* holds in the situation resulting from executing *DrawCard* in s.

Previously we failed to arrive at the conclusion that the agent had a cup of coffee after the performance of the "draw card"-"wait"-"press button" sequence, and we now fail to infer the sentence

$$\begin{aligned}&Holds(\textit{has-coffee}, result(\textit{PressButton}, \\ &\quad result(\textit{Wait}, result(\textit{DrawCard}, S0)))).\end{aligned} \tag{2.4}$$

The reason is that although we can infer that

$$Holds(\textit{credit}, result(\textit{DrawCard}, S0)) \tag{2.5}$$

there is nothing that lets us infer

$$Holds(credit, result(Wait, result(DrawCard, S0))) \tag{2.6}$$

which is required in order to infer (2.4).

What is needed is a logical version of the rule that fluents remain the same unless explicitly caused to change by an action. One way of formulating this is with explicit frame axioms. A frame axiom states that a specific fluent remains true or false when a specific action is executed. The following are the frame axioms that would need to be added to (2.1–2.3).

$$\begin{array}{l}
\forall s[Holds(credit, s) \Rightarrow Holds(credit, result(Wait, s))] \\
\forall s[\neg Holds(credit, s) \Rightarrow \neg Holds(credit, result(Wait, s))] \\
\forall s[\neg Holds(credit, s) \Rightarrow \\
\quad \neg Holds(credit, result(PressButton, s))] \\
\forall s[Holds(has\text{-}coffee, s) \Rightarrow \\
\quad Holds(has\text{-}coffee, result(DrawCard, s))] \\
\forall s[\neg Holds(has\text{-}coffee, s) \Rightarrow \\
\quad \neg Holds(has\text{-}coffee, result(DrawCard, s))] \\
\forall s[Holds(has\text{-}coffee, s) \Rightarrow \\
\quad Holds(has\text{-}coffee, result(Wait, s))] \\
\forall s[\neg Holds(has\text{-}coffee, s) \Rightarrow \\
\quad \neg Holds(has\text{-}coffee, result(Wait, s))] \\
\forall s[Holds(has\text{-}coffee, s) \Rightarrow \\
\quad Holds(has\text{-}coffee, result(PressButton, s))] \\
\forall s[\neg Holds(has\text{-}coffee, s) \wedge \neg Holds(credit, s) \Rightarrow \\
\quad \neg Holds(has\text{-}coffee, result(PressButton, s))]
\end{array} \tag{2.7}$$

2.5.2 Problems with frame axioms

Unfortunately, the use of frame axioms has a number of drawbacks, and these drawbacks are not restricted to the frame problem but are also closely connected to the ramification and concurrency problems.

First, an explicit enumeration such as in (2.7) involving almost every action-fluent-pair will require about $2 \cdot m \cdot n$ frame axioms if there are m action types and n fluent types. Thus, the use of explicit frame axioms is very space-inefficient, and the large amount of axioms would make theorem proving less efficient. Yet, if most of the frame axioms can be produced automatically, one can argue that explicit frame axioms are still manageable. At least for the type of actions in the example above, this appears to be the case.

A more serious problem arises if the relation between the initial state of an action and its effects is more complex, as in the presence of ramifications, or side-effects, as discussed in section 2.6. It is desirable not to have to specify all ramifications of an action in the description of that particular action. For instance, assume that we add one more fluent *cup-broken* representing the fact that the coffee cup is broken. If the cup breaks, the coffee is lost, as the following constraint states:

$$\forall s[Holds(\textit{cup-broken}, s) \Rightarrow \neg Holds(\textit{has-coffee}, s)] \tag{2.8}$$

Now, if we encode frame axioms for *has-coffee*, we have to take this constraint into consideration and make sure that whenever an action makes *cup-broken* true, no frame axioms for *has-coffee* should be applicable. When the specification of change becomes more complex, so does the specification of no-change. Consequently, it is no longer sufficient to consider what fluents are involved in direct effects when writing frame axioms, but also indirect effects have to be taken into account.

Another serious problem arises if actions can be concurrent. Combining the frame axioms of two concurrent actions will in most cases result in inconsistency. For instance, if the frame and effect axioms for $(DrawCard||Wait)$ (drawing the card and waiting in concurrency) are derived by taking the conjunction of the frame and effect axioms for $DrawCard$ and $Wait$, the resulting action description can contain contradictions under some conditions. The fluent *credit* has the following axioms.

$$\begin{aligned}
&\forall s[\, Holds(\textit{credit}, result((DrawCard||Wait), s)) \wedge \\
&\quad (Holds(\textit{credit}, s) \Rightarrow \\
&\qquad Holds(\textit{credit}, result((DrawCard||Wait), s))) \wedge \\
&\quad (\neg Holds(\textit{credit}, s) \Rightarrow \\
&\qquad \neg Holds(\textit{credit}, result((DrawCard||Wait), s)))\,].
\end{aligned}$$

If $\neg Holds(\textit{credit}, S0)$ is true, then there is a contradiction between

$$Holds(\textit{credit}, result((DrawCard||Wait), S0))$$

from the first conjunct, and

$$\neg Holds(\textit{credit}, result((DrawCard||Wait), S0))$$

from the last conjunct. Thus, concurrent action combinations such as

$$(DrawCard||Wait)$$

require their own sets of frame axioms, and as the number of concurrent action combinations is so much larger than the number of single actions, the number of frame axioms required would grow drastically.

2.5.3 Aspects of the frame problem

We should also mention that the term "the frame problem" has been used in a number of different ways, and can be defined on different levels of generality. Morgenstern [106] provides an interesting enumeration of different versions of the frame problem, and also discusses criteria for evaluating different solutions.

In conclusion, there are several aspects to the frame problem. One aspect concerns specifying what does not change (or, more generally, is not affected), which becomes increasingly more difficult the more advanced the ontology of the world is. Another aspect concerns how to construct this specification in a compact way. The method of explicit frame axioms is unsatisfactory in both these respects.

2.6 The ramification problem

The ramification problem is closely related to the frame problem. While the frame problem concerns what does not change, the ramification problem concerns what actually does change due to an action. As an example, consider the action of pressing the button again. The specification of the action was very simplified: the only result of the action was that the agent got coffee and the credit was lost. However, one can imagine a number of other possible effects, of a more indirect nature. One example was the relation between *has-coffee* and *cup-broken* in (2.8). In addition, if a coffee cup is broken, the coffee might be spilled on a carpet and/or on the agent itself, and the carpet gets stained and the agent (if human) gets burned, and so on. In summary, the execution of an action can trigger arbitrarily long chains of causes and effects. The problem of specifying all these effects in a compact manner is called the ramification problem. In particular, it is desirable not to have to declare all potential indirect effects in the axioms for specific actions.

As mentioned above, the more complex the specification of changes, the more complex it becomes to specify what does not change. Thus, the frame and ramification problems are not two problems that can be solved in isolation, but two problems that require a solution in common.

2.7 The qualification problem

The qualification problem is the last of the classic triple, and concerns the specification of all the conditions that can cause an action to fail, or otherwise to produce different results from what is intended or expected. In the context of pressing the button of the coffee vending machine, for instance, the normal effects might be prevented by the fact that the machine has run out of coffee powder, or the electricity is switched off, or the water has frozen, or innumerable other conditions.

Here, there are two aspects to the problem that require consideration. First, as the number of conditions that can make an action fail can be very large, it is desirable not to have to put all these conditions into one single law (axiom) describing the action in question. Instead, it would be preferable if one could separate these conditions from the rest of the action description, and if one could add them in an incremental manner. Ideally, one should be able to have one law or axiom for each condition, and then combine them automatically, for instance by assuming that these conditions are the only conditions that can make the action fail.

The second aspect of the problem is of a more complex nature. Although one has described that pressing the button will not produce any coffee if the machine has run out of coffee powder, or the electricity is switched off, or the water has frozen, etc, it still remains to determine whether any of these conditions actually holds. Determining this is not just a matter of making suitable inferences, but might require the agent to make observations, take measurements, and even set up experiments — altogether a very time-consuming procedure. Consequently, it might be more efficient to let the agent jump to the conclusion that none of these condition holds, and attempt to execute the action. If the action yields unexpected results, then the agent can revise its previous conclusions about these conditions and start reasoning about what could have gone wrong. Yet, how to formalize these inferential jumps ("there is enough coffee powder, etc") is a non-trivial problem.

2.8 The concurrency problem

In addition to the three problems mentioned above, there is a fourth problem that is of equal significance for RAC. Most of the work in RAC has been done under the assumption that there is one single agent in the world performing actions sequentially. If this assumption is relaxed, then actions of different

agents, or of the same agent, might occur concurrently. In the simplest case, the combined effects of two (or more) concurrent actions is the union of the individual effects. As we have seen, this would presuppose a solution to the frame problem that, unlike frame axioms, permits us to combine the effects of multiple actions. There is also the possibility that actions might interact, i.e. the effects of a number of concurrent actions might be different from the union of the effects each action would have had if they had been executed in isolation. How to represent such interactions in a graceful manner, in particular without having to explicitly consider each possible combination of concurrent actions, is one of the central issues of this thesis (paper I).

2.9 Summary

In this chapter, we have presented some of the problems addressed in reasoning about action and change. We have considered a number of aspects of representations for action and change, ranging from ontological consideration, to epistemological and higher-level ones. We have presented a number of problems encountered in the design of action and change representations: the frame problem, the ramification problem, the qualification problem and the concurrency problem. In particular, we observed that the frame problem is intrinsically related to the ramification and concurrency problems. The experience that the different problems of reasoning about action and change and their prospective solutions are strongly connected to many of the other problems will recur throughout this thesis, in particular in paper I where we see how dealing with concurrent actions cannot be isolated from dealing with ramifications, and in paper II where we see how dealing with delayed effects of actions requires dealing with both ramifications and concurrency. In the next chapter, we proceed to investigate a number of existing representations and how and to what extent they address these problems.

Chapter 3

Languages for reasoning about action and change

In this chapter, we present a number of languages for reasoning about action and change. The purpose of this survey is two-fold. First, it is intended to give the reader a general overview of the area of RAC and the space of solutions that have been explored. Second, several of the languages presented here will play important roles in the discussions in the subsequent chapters and papers. The survey does not attempt to be completely exhaustive: instead it focuses on the classes of languages that most frequently occur in the RAC literature. It does not include RAC-related work that goes beyond reasoning about actions and their external effects; consequently, results from for instance action and knowledge [105, 79], robot programming [78, 24, 121] and goal-oriented behavior [125] are not included.

We begin with the situation calculus (SC), which is a class of languages based on classical logic with a branching event-driven temporal structure [97, 118]. Each branch represents a hypothetical execution of an action sequence. Thus, a situation calculus theory can be viewed as representing all alternative courses of actions that are possible. The main advantage of the situation calculus is the fact that action sequences (situations) are first-order objects, a fact that supports deductive reasoning about alternative choices of actions. We also address the language $\mathcal{A}$ [43], and the fluent calculus [55, 18], which are based on assumptions similar to those of the situation calculus.

There is another group of languages that can represent one actual course of events as opposed to the multiple hypothetical courses of events represented in SC; we call these languages narrative-based. This has both ad-

vantages (e.g. for representing and reasoning about observations) and disadvantages (reasoning about alternative courses of events has to be done in terms of alternative logical theories). Narrative-based languages are typically based on a notion of time that is independent of the execution of actions. Consequently, such languages have the advantage that they support richer temporal properties and relations than languages with branching event-driven time. Some examples of narrative-based languages are action logics based on Allen's temporal interval algebra [6], Shoham's logic of chronological ignorance [132], Morgenstern's and Stein's motivated action theory [107], the logics by Kautz [67] and Haugh [50], and McCain's and Turner's causal theories [93]. Here, we focus on the two most widely studied narrative-based languages, namely the event calculus (EC) [69, 129] and the temporal action logic (TAL) [124, 26, 29]. In addition, we present the language $\mathcal{L}_1$ [13] which combines some of the features of branching and narrative-based languages.

Finally, we also address some modal logics, namely dynamic logic [116], and CTL* [21]. Although they are less frequently used in RAC, these logics represent interesting alternative perspectives on time, actions and developments.

3.1 Situation calculus

The situation calculus, originally due to McCarthy and Hayes [97][1], is one of the oldest and arguably the most widely studied and used formalisms of action and change. In fact, the situation calculus is not one uniform language, but exists in many variations. The presentation here relies mainly on the work by Reiter [118], but with some small differences that will be pointed out.[2] We denote this version of SC with SC(R). There are other versions of situation calculus which differ from Reiter's in important respects,

[1] McCarthy's and Hayes's version of the situation calculus is significantly different from the later versions presented in this section. For instance, the *result* function, which defines the situation resulting from executing an action, has three arguments: the agent that performs the action, the action itself, and the situation in which the action is executed. Actions can be combined into strategies, which include pure sequences of actions, but also choices, loops etc. Situations are viewed as "snapshots" of the world; there is not the one-to-one correspondence between situations and action sequences which is present in the work of Reiter. There are also discussions about modal operators for defeasible reasoning ($normally(\Theta), consistent(\Theta), probably(\Theta)$) and about knowledge and abilities.

[2] These differences will not matter in the discussions about the strengths and weaknesses of the situation calculus.

for instance the situation calculus for combining narratives presented by McCarthy and Costello [96].

SC(R) is based on many-sorted predicate calculus with equality. The following elements are central.

- A sort $\mathcal{S}$ for situations, with variables s, s', s_1 etc. The constant $S0$ represents the initial situation.
- A sort $\mathcal{A}$ for actions, with variables a, a', a_1 etc.
- A sort $\mathcal{F}$ for fluents, with variables f, f', f_1 etc.
- The predicate $Holds : \mathcal{F} \times \mathcal{S}$ associates each situation with a state, in which each fluent has a truth value.
- The function $result : (\mathcal{A} \times \mathcal{S}) \rightarrow \mathcal{S}$ returns the situation resulting from applying an action in a situation.
- The predicate $Poss : \mathcal{A} \times \mathcal{S}$ represents whether it is possible to execute an action in a situation.

The fact that fluents are terms and the use of a $Holds$ predicate are due to a technique called *reification*. An advantage of reification is that it is possible to quantify over fluents. A common alternative, used by McCarthy-Hayes [97] and Reiter [118] among others, is to represent fluents by predicates with a situational argument, and denote the fact that f holds in s by $f(s)$. Using a reified approach, the same fact is expressed as $Holds(f, s)$.

The $result$ function forms the temporal structure of SC(R). A number of applications $result(a_n, \ldots result(a_1, s) \ldots)$ represents the situation after the execution of a sequence of actions $[a_1, \ldots, a_n]$ in the situation s. Some people, for instance Reiter [118], call the function do instead of $result$. In Reiter's situation calculus, the actions in the sequence are assumed to be the only actions occurring.

McCarthy and Costello [96], who address the problem of representing and combining narratives, have a significantly different approach. Instead of relying on terms of the form $result(a_n, \ldots result(a_1, s) \ldots)$ for referring to action sequences, they introduce a special predicate $Occurs : \mathcal{A} \times \mathcal{S}$ for action occurrences (e.g. $Occurs(Does(robot, DrawCard), S1)$) and they use explicit ordering constraints between situations (e.g. $S1 < S2$). This representation permits other actions to occur in between or concurrently with those explicitly stated in the narrative, and hence it is possible to combine two narratives into one, in addition to other operations.

3.1.1 A tree of situations

A natural way to view the temporal structure of SC(R) is as an infinite tree of situations with $S0$ as root, where each finite (infinite) branch corresponds to a finite (infinite) sequence of actions, and different branches represent different choices of action sequences. Each situation has a unique temporal identity based on the action sequence that leads to it. Two situations s_1 and s_2 might represent the same state ($\forall f[Holds(f, s_1) \equiv Holds(f, s_2)]$) but still be two distinct objects ($s_1 \neq s_2$) as they correspond to different action sequences. For instance, the situations

$$result(Wait, result(DrawCard, S0))$$

and

$$result(DrawCard, result(Wait, S0))$$

would be distinct although their associated states would be equivalent. There might also be states that are not associated with any situation.

The view of situations as nodes in a tree requires three *foundational axioms*[3].

$$\forall a, s[result(a, s) \neq S0] \tag{3.1}$$

$$\forall a_1, a_2, s_1, s_2[\, result(a_1, s_1) = result(a_2, s_2) \Rightarrow (a_1 = a_2 \wedge s_1 = s_2)] \tag{3.2}$$

$$\forall P[(P(S0) \wedge \forall a, s[P(s) \Rightarrow P(result(a, s))]\,) \Rightarrow \forall s[P(s)]\,] \tag{3.3}$$

The first two axioms (3.1) and (3.2) ensure that there are no loops or merges in the tree structure. The last axiom is a second-order inductive axiom (P is a second-order variable representing the property being quantified over). This last axiom (3.3) defines the universe of situations to be the smallest set that can be constructed with the initial situation $S0$ and the *result* function. In addition, it can be applied in proofs of properties holding in all situations, which is useful for instance when proving database integrity constraints and non-existence of plans for a specific goal [119].

Given the tree structure, it is meaningful to introduce a precedence ordering $<$ on situations:

$$\forall s[\neg s < S0] \tag{3.4}$$

$$\forall s, s', a[s < result(a, s') \equiv (Poss(a, s') \wedge (s = s' \vee s < s'))] \tag{3.5}$$

[3]The axioms presented here are from [118]. The two first axioms appear in a slightly different form in [119].

3.1.2 Situations as states

An alternative way is to view situations as states, with no separate temporal identity, as done by Baker [9]. It is assumed that there is at most one situation for each state.

$$\forall f, s_1, s_2[Holds(f, s_1) \equiv Holds(f, s_2) \Rightarrow s_1 = s_2] \tag{3.6}$$

In addition, Baker's approach, due to the particular non-monotonic policy employed, requires a second-order state existence axiom. It is not clear whether this axiom would be required for every state-based approach.

$$\forall P \exists s \forall f[P(f) \equiv Holds(f, s)] \tag{3.7}$$

This axiom states that for each assignment of truth values to fluents, that is each state (the second-order variable P), there is an equivalent situation s. In short, there is at least one situation for each state. Together, (3.6) and (3.7) asserts a one-to-one correspondence between states and situation, making the two concepts equivalent.

3.1.3 Actions and their effects

The effects of actions can be specified in axioms as below.

$$Poss(\mathbf{a}(\overline{x}), s) \wedge P(\overline{x}, \overline{y}, s) \Rightarrow R(\overline{x}, \overline{y}, result(\mathbf{a}(\overline{x}), s)) \tag{3.8}$$

The expressions $P(\overline{x}, y, s)$ and $R(\overline{x}, y, result(\mathbf{a}(\overline{x}), s))$ denote context and effect specifications where the $Holds$ predicate only refers to the situations s and $result(\mathbf{a}(\overline{x}), s)$, respectively. In the case where the action $\mathbf{a}(\overline{x})$ is context-dependent, there is one axiom for each context-effect pair. The following is an example (all free variables are considered universally quantified).

$$Poss(DrawCard, s) \Rightarrow Holds(credit, result(DrawCard, s)) \tag{3.9}$$

$$\begin{aligned} &Poss(PressButton, s) \wedge Holds(credit, s) \Rightarrow \\ &\quad (Holds(has\text{-}coffee, result(PressButton, s)) \wedge \\ &\quad \neg Holds(credit, result(PressButton, s))) \end{aligned} \tag{3.10}$$

The $Poss$ predicate, which determines when actions are executable, also needs to be specified.

$$Holds(has\text{-}card, s) \Rightarrow Poss(DrawCard, s) \tag{3.11}$$

$$T \Rightarrow Poss(PressButton, s) \tag{3.12}$$

3.1.4 A simple solution (sometimes) to the frame problem in SC

If action effects are assumed to be deterministic, and thus can be written as conjunctions of literals, an alternative is to write action effect axioms in the form below. This form enables an elegant solution to the frame problem for some classes of worlds [118]. Notice that the action component is implicitly quantified over.

$$\begin{array}{l} Poss(a,s) \wedge \alpha_f(a,\overline{x},\overline{y},s) \Rightarrow Holds(\mathbf{f}(\overline{x}), result(a,s)) \\ Poss(a,s) \wedge \delta_f(a,\overline{x},\overline{y},s) \Rightarrow \neg Holds(\mathbf{f}(\overline{x}), result(a,s)) \end{array} \tag{3.13}$$

The expression $\alpha_f(a,\overline{x},\overline{y},s)$ specifies the conditions and actions under which $\mathbf{f}(\overline{x})$ will become true, and the expression $\delta_f(a,\overline{x},\overline{y},s)$ denotes the actions and conditions under which $\mathbf{f}(\overline{x})$ will become false. The former can be derived by taking the conjunction of all action and context combinations (the $P(\overline{x},\overline{y},s)$ parts in (3.8)) where $Holds(\mathbf{f}(\overline{x}), result(a,s))$ occurs as positive in the consequent, and the latter from the action and context combinations where $Holds(\mathbf{f}(\overline{x}), result(a,s))$ occurs negated. For the axioms (3.9) and (3.10), the equivalent form is as follows.

$$Poss(a,s) \wedge a = DrawCard \Rightarrow Holds(credit, result(a,s)) \tag{3.14}$$

$$\begin{array}{l} Poss(a,s) \wedge (a = PressButton \wedge Holds(credit,s)) \Rightarrow \\ \quad \neg Holds(credit, result(a,s)) \end{array} \tag{3.15}$$

$$\begin{array}{l} Poss(a,s) \wedge (a = PressButton \wedge Holds(credit,s)) \Rightarrow \\ \quad \neg Holds(\textit{has-coffee}, result(a,s)) \end{array} \tag{3.16}$$

If one assumes that these axioms specify exactly the conditions under which f becomes true respectively false (Reiter's completeness assumption [118]), one can generate *successor state axioms* [118].[4] A successor state axiom specifies exactly when a specific individual feature is true in a situation in terms of what holds in the preceding situation and what action has been performed. The general schema for a successor state axiom is as below,

$$\begin{array}{l} Poss(a,s) \Rightarrow \\ \quad [Holds(\mathbf{f}(\overline{x}), result(a,s)) \equiv \\ \qquad (\alpha_f(a,\overline{x},\overline{y},s) \vee (\neg\delta_f(a,\overline{x},\overline{y},s) \wedge Holds(\mathbf{f}(\overline{x}),s)))]. \end{array} \tag{3.17}$$

[4] The idea of successor state axioms has its roots in the works of Pednault [108], Haas [49] and Schubert [126].

For instance, the successor state axioms corresponding to (3.14–3.16) are as follows, somewhat simplified,

$$\begin{aligned} &Poss(a,s) \Rightarrow [Holds(credit, result(a,s)) \equiv \\ &\quad (a = DrawCard \vee \\ &\quad ((\neg a = PressButton \wedge Holds(credit, s)))))] \end{aligned} \tag{3.18}$$

$$\begin{aligned} &Poss(a,s) \Rightarrow [Holds(has\text{-}coffee, result(a,s)) \equiv \\ &\quad F \vee (\neg(a = PressButton \wedge Holds(credit, s)) \\ &\quad \wedge Holds(has\text{-}coffee, s))]. \end{aligned} \tag{3.19}$$

Observe how the $Poss$ predicate is used in (3.17): if $Poss(a,s)$ does not hold, no conclusions can be made about whether $\mathbf{f}(\overline{x})$ holds in the next situation. Consequently, the result of a non-executable action is totally unspecified. It can be any situation.

In summary, Reiter's approach offers an efficient solution to the frame problem in SC(R) for a limited class of worlds. The completeness assumption is quite strong, and rules out nondeterministic action effects as these cannot be expressed in the form of (3.13). Among the advantages of the approach are that the generation of successor state axioms can be done automatically, and that successor state axioms are useful when using regression techniques.

3.1.5 On situations as first-order objects

In SC(R), situations are first-order objects that correspond to action sequences executed from the initial situation $S0$. Thus, action sequences can be considered reified and can be quantified over. However, not all situations need correspond to the successful execution of an action sequence[5]. In order to distinguish these "ghost situations" from situations corresponding to action sequences executable in $S0$, a predicate $ex : \mathcal{S}$ (for "executable") is defined as follows.

$$\begin{aligned} &ex(s) \equiv \\ &\quad (s = S0 \vee \exists a, s'[s = result(a, s') \wedge Poss(a, s') \wedge ex(s')]) \end{aligned} \tag{3.20}$$

That situations are first-order objects makes it possible to reason about different action sequences simply by referring to different situations. Thus, SC(R) supports hypothetical reasoning.

[5]Note that in the state-based approach, it can also be the case that a situation corresponds to several action sequences.

It is also possible to some extent to describe operations such as merging together two (or more) sets of action occurrences. In SC(R), each set (or rather sequence) of actions corresponds to a situation

$$result(a_n, \ldots result(a_1, s)\ldots)$$

where s represents the starting situation of the sequence. In section 2.3 in chapter 2, we gave the example "if the actions in A lead to α, and the actions in A' lead to β given that α holds in the beginning, then A and A' together, in sequence, lead to β". In SC(R), the statement "the action sequence $a_1, \ldots, a_n$ leads to α" can be formulated

$$\forall s[Holds(\alpha, result(a_n, \ldots result(a_1, s)\ldots))],$$

and "if α holds, then the sequence $a'_1, \ldots, a'_n$ leads to β" can be formulated

$$\forall s[Holds(\alpha, s) \Rightarrow Holds(\beta, result(a'_n, \ldots result(a'_1, s)\ldots))].$$

Finally, "$a_1, \ldots, a_n$ and $a'_1, \ldots, a'_n$ together, in sequence, lead to β" can be formulated

$$\begin{aligned}&\forall s, s'[s' = result(a_n, \ldots result(a_1, s)\ldots) \Rightarrow \\ &\quad Holds(\beta, result(a'_n, \ldots result(a'_1, s')\ldots))].\end{aligned}$$

Observe that this concerns the merging of specific action sequences; it is not possible to refer to the merging of action sequences in general, and to state for instance that "for any two sequences, if the first one leads to α, and the second one leads to β provided α holds, then the two sequences together lead to β." The reason is that action sequences cannot be decoupled from their starting situations. A term representing an action sequence

$$result(a_n, \ldots result(a_1, s)\ldots)$$

always includes some starting situation s. Consequently, one cannot refer to just the action sequence $[a_1, \ldots, a_n]$ as an independent object.

An appealing feature of SC(R) is that plan synthesis problems can be specified in a straightforward manner. Let $G(s)$ be a description of a goal: a logical combination of *Holds* statements referring to the situation variable s. The initial situation $S0$ and the preconditions and effects of actions are assumed to be appropriately axiomatized. The problem to find a plan (sequence of actions) that leads from the initial situation to a situation satisfying the goal is equivalent to proving the following sentence.

$$\exists s[exec(s) \wedge G(s)] \tag{3.21}$$

The s that is found to satisfy the condition above, and which is generated as a side effect of the existence proof, is a plan. Thus, planning becomes a purely deductive problem. We will soon compare this to the plan synthesis problem in narrative-based approaches, where it becomes abductive. The fact that plan synthesis can be done deductively in SC(R) does not imply that actually finding a plan need be any easier than in other settings, though. It has been argued by, for instance, Reiter[120] that deductive planning avoids the need for consistency checking of plans, but, as Miller [101] has pointed out, this can also be avoided in an abductive setting if plans are constrained to always be totally ordered and ground. It should also be recognized that the definition of the planning problem above has limitations; in particular, it presupposes a completely specified initial situation and deterministic action effects. This problem will be further addressed in paper IV.

Yet, being able to treat entities such as action sequences (which start at specific situations) as first-order objects is a matter of great interest from the perspective of expressiveness. It widens the range of concepts and ideas that can be formalized and analyzed within the formal language in use. Therefore, it is one of the major issues of this thesis.

3.1.6 Situation calculus: advanced ontologies

The versions of SC presented so far have been quite restricted from an ontological perspective. For instance, they do not support nondeterministic effects of actions or ramifications.

Nondeterminism and ramifications have been addressed by Lin [88, 87]. Lin introduces a special predicate $Caused(f, s, v)$ representing the fact that a fluent f is caused to obtain the boolean value v in situation s. For instance, $Caused(credit, result(DrawCard, s), true)$ represents that drawing the card through the card reader gives the agent a credit in the coffee machine. Consequently, the coffee machine scenario can be specified as follows,

$$\begin{aligned} &Poss(DrawCard, s) \Rightarrow \\ &\quad Caused(credit, result(DrawCard, s), true) \end{aligned} \tag{3.22}$$

$$\begin{aligned} &Poss(PressButton, s) \wedge Holds(credit, s) \Rightarrow \\ &\quad (Caused(has\text{-}coffee, result(PressButton, s), true) \wedge \\ &\quad\ Caused(credit, result(PressButton, s), false)). \end{aligned} \tag{3.23}$$

The relation between $Caused$ and $Holds$ is central to Lin's approach. Whenever a fluent is caused to be true or false, it will also hold/not hold.

$$Caused(f, s, true) \Rightarrow Holds(f, s) \tag{3.24}$$

$$Caused(f, s, false) \Rightarrow \neg Holds(f, s) \tag{3.25}$$

The assumption is that only features that occur in *Caused* statements are allowed to change. One minimizes *Caused*, and then one filters with the statement that fluents that are not involved in *Caused* do not change:[6]

$$\begin{array}{l} \neg\exists v[\, Caused(f, result(a, s), v)\,] \Rightarrow \\ \quad (Holds(f, s) \equiv Holds(f, result(a, s))). \end{array} \tag{3.26}$$

As it is possible to have disjunctions of *Caused* statements, nondeterministic effects can be represented. The following example describes the action of tossing a coin.

$$\begin{array}{l} (Caused(heads, result(Toss, s), true) \wedge \\ \; Caused(tails, result(Toss, s), false)) \vee \\ (Caused(heads, result(Toss, s), false) \wedge \\ \; Caused(tails, result(Toss, s), true)) \end{array} \tag{3.27}$$

Furthermore, a *Caused* statement need not be associated with any specific action. This makes Lin's approach suitable for representing ramifications, which are changes that are indirectly caused by actions. An indirect effect (ramification) can be specified as in the following example, which describes that if the cup breaks, then you lose your coffee.

$$Holds(cup\text{-}broken, s) \Rightarrow Caused(has\text{-}coffee, s, false) \tag{3.28}$$

The ramification problem has also been addressed by McIlraith [99] who suggests encoding ramifications into the successor state axioms of fluents (see section 3.1.4). McIlraith uses regression operators to automatically compile effect axioms and ramification constraints into a set of successor state axioms. For instance, assume that the following effect axioms and ramification constraints hold.

$$\forall s[Holds(cup\text{-}broken, s) \Rightarrow \neg Holds(has\text{-}coffee, s)] \tag{3.29}$$

$$\forall s[Holds(cup\text{-}broken, result(DropCup, s))] \tag{3.30}$$

Somewhat simplified, the successor state axiom for *has-coffee* is generated as follows. First, one computes an intermediate successor state axiom. The only circumstance under which *has-coffee* can change is described in (3.29),

[6] As observed by Gustafsson and Doherty [48], this is the same principle as in Sandewall's PMON logic [124] and TAL (see section 3.6), where occlusion serves a purpose similar to *Caused* in Lin's formalism.

that is to say when the cup is broken. Consequently, the intermediate successor state axiom for *has-coffee* is as follows.

$$\begin{array}{l}\forall s, a[Poss(a, s) \Rightarrow (Holds(\textit{has-coffee}, result(a, s)) \equiv \\ \quad (Holds(\textit{has-coffee}, s) \wedge \\ \quad \neg Holds(\textit{cup-broken}, result(a, s))))].\end{array} \tag{3.31}$$

The agent has coffee if and only if it had coffee in the previous situation, and the cup is not broken in the current situation.

One can now regress over $Holds(\textit{cup-broken}, result(a, s))$, in order to obtain the conditions under which this statement is true (and which indirectly makes *has-coffee* false). The (final) successor state axiom for *cup-broken* is

$$\begin{array}{l}\forall s, a[Poss(a, s) \Rightarrow (Holds(\textit{cup-broken}, result(a, s)) \equiv \\ \quad (a = \textit{DropCup} \vee Holds(\textit{cup-broken}, s)))].\end{array} \tag{3.32}$$

Consequently, one can replace $Holds(\textit{cup-broken}, result(a, s))$ in (3.31) with the right-hand-side of the equivalence in (3.32):

$$\begin{array}{l}\forall s, a[Poss(a, s) \Rightarrow (Holds(\textit{has-coffee}, result(a, s)) \equiv \\ \quad (Holds(\textit{has-coffee}, s) \wedge \\ \neg(a = \textit{DropCup} \vee Holds(\textit{cup-broken}, s))))].\end{array} \tag{3.33}$$

Equation (3.33) is called the final successor state axiom for *has-coffee*. In summary, McIlraith's approach makes elegant use of regression in order to "compile away" ramification constraints. A limitation of the approach is that the ramification constraints may not contain any cyclic dependencies.

Finally, the topic of domain constraints (without explicit causal directionality) has been explored by (among others) Lin and Reiter [89], although their solution was not based on successor state axioms. A number of other advanced issues pertaining to the situation calculus will be addressed in chapters 4 and 5, namely concurrency, explicit time, natural events and triggered actions.

3.1.7 Discussion

Fluents (when reified), actions and situations/action sequences are all first-order objects in SC(R). The fact that action sequences are terms distinguishes SC(R) from the narrative-based formalisms in the latter part of this chapter, and this feature has been used to justify SC(R) versus other RAC formalisms [120]. It is possible to reason about alternative action sequences, and even to quantify over action sequences in SC(R).

In SC(R), though, there is no independent notion of time; the progression of time is completely dependent on the execution of actions. There are no means for expressing that a fact holds or an action or event occurs at some specific moment or over some specific period, measured in for instance seconds, minutes, hours, days, or some other metric unit. This leads to a number of difficulties, some of which will be briefly addressed here. As actions are the basis of the temporal structure, there are no means to efficiently relate actions temporally in non-trivial ways such as when the order among actions is incompletely known (partial orders) or that actions are partially overlapping. Further, as the *Holds* predicate relates to what is true in situations, and actions are transitions *between* situations, there are no means to express what holds during the execution of an action.

Finally, each action sequence is only associated with one single situation. Therefore, it is not possible to make statements about alternative consequences of an action sequence. For instance, the statement "the action sequence $[a_1, \ldots, a_n]$ always/necessarily has the effect α" is formulated in terms of logical consequence:

$$\Gamma \models Holds(\alpha, result(a_n, \ldots result(a_1, s)\ldots)).$$

The statement "the action sequence $[a_1, \ldots, a_n]$ sometimes/possibly has the effect α" is formulated

$$\Gamma \not\models \neg Holds(\alpha, result(a_n, \ldots result(a_1, s)\ldots)).$$

In paper IV in this thesis, we show how this limitation affects the possibility to plan deductively with SC(R), in particular in the presence of incomplete initial information and nondeterministic effects. There, we also present a modified SC with the following properties:

- Action sequences are objects that are distinct from situations.
- Each action sequence corresponds to a set of developments. Consequently, it is possible to make logical statements about necessary and possible consequences of an action sequence.
- Nondeterministic effects of actions can be represented.
- It possible to plan purely deductively also in the presence of incomplete initial information and nondeterministic effects.

3.2 Fluent calculus

The fluent calculus (FL) [55, 18] is a language with some strong similarities to the situation calculus. The distinguishing feature of the fluent calculus is that states are reified. Essentially, a state is represented as a term composed by those fluents that are true in the state. State terms are composed using the $\circ$ operator, like in *credit* $\circ$ *has-coffee*. The temporal structure is the same as for situation calculus: a tree of situations, starting with an initial situation $S0$ and then extended by applying actions ($Do(a, s)$). The function $State(s)$ associates each situation with a specific state.

State update axioms relate the state before an action to the state after the action in terms of equality (instead of logical implication and equivalence as in SC(R)). For instance, the following is a state update axiom for drawing the card through the card reader.

$$\begin{aligned} &Poss(DrawCard, s) \Rightarrow \\ &\quad [State(Do(DrawCard, s)) = State(s) \circ credit] \end{aligned} \tag{3.34}$$

In short, this axiom states that the state after the action is the same as the state before the action, but with *credit* added. The $\circ$ operator is commutative and associative, so the order in which fluents appear does not matter. Likewise, fluents can easily be removed from a state, like *credit* in the following state update axiom for pressing the button of the coffee machine.

$$\begin{aligned} &Poss(PressButton, s) \Rightarrow \\ &\quad [State(Do(PressButton, s)) \circ credit = \\ &\quad\ State(s) \circ has\text{-}coffee] \end{aligned} \tag{3.35}$$

Notice that in the fluent calculus, entire state updates are made using one single axiom. Thereby, state updates can be made more efficiently than in for instance SC(R), where each fluent requires the use of a separate axiom (see section 3.1.4). This is one of the main motivations behind the fluent calculus. Nondeterministic actions can be described using disjuncts of equalities in state update axioms. Ramifications (indirect effects) are realized by introducing an additional predicate

$$Causes(state, effects, new\_state, new\_effects)$$

that specifies sequences (cascades) of state updates following an action [136].

The main difference between the fluent calculus and SC(R) is the notion of a state: in FL states are objects, whereas in SC(R) they are defined through the *Holds* predicate (or other predicates with a situational argument). The different notions of state also imply different ways of describing

state transitions; in FL, these descriptions become more concise than in SC(R). The temporal structure is the same in the two formalisms: branching event-driven time. Consequently, the comments made in section 3.1.7 — about alternative action sequences, the lack of an independent notion of time and the fact that each action sequence is associated with one single situation — also apply to the fluent calculus.

3.3 $\mathcal{A}$-style languages

In the early 90s, the need for methodologies that allowed systematic evaluations of and comparisons between formalisms for RAC was recognized. One influential contribution was Sandewall's book *Features and Fluents* [124], another was the language $\mathcal{A}$ by Gelfond and Lifschitz [43] and its successors [138, 11, 81, 18].

The language $\mathcal{A}$ is intended as a concise high-level language with no-change and other ontological choices encoded into its operational semantics in a manner similar to the STRIPS solution [39]. A particular logic for RAC can then be described as a translation from $\mathcal{A}$, or a subset or superset of $\mathcal{A}$, to for instance a theory of classical logic or a logic program. The temporal ontology of $\mathcal{A}$ is based on the same type of branching, event-driven time as SC(R), and the language and its successors have mainly been used for evaluating existing SC versions and proposing new ones (the latter are typically based on some form of logic programming). The fluent calculus is another language to which the $\mathcal{A}$ methodology has been applied [18].

A domain description in $\mathcal{A}$ consists of value propositions of the form "**initially** F" denoting that F holds before any actions have been executed, and effect propositions of the form "A **causes** F **if** $P_1, \ldots, P_n$" denoting that if A is executed in a state where the preconditions (literals) $P_1, \ldots, P_n$ hold, then F holds in the subsequent state. The following is an example.

$$
\begin{array}{ll}
1 & \textbf{initially } \neg \textit{credit}. \\
2 & \textbf{initially } \neg \textit{has-coffee}. \\
3 & \textit{DrawCard } \textbf{causes } \textit{credit}. \\
4 & \textit{PressButton } \textbf{causes } \textit{has-coffee } \textbf{if } \textit{credit}. \\
5 & \textit{PressButton } \textbf{causes } \neg \textit{credit}.
\end{array}
\tag{3.36}
$$

There are also value propositions of the form "F **after** $A_1; \ldots; A_m$" which denotes that the fluent literal F holds after the execution of the action sequence $A_1; \ldots; A_m$. These can both be used in domain descriptions and as queries about a domain description. For instance, given (3.36), the query "*has-coffee* **after** *DrawCard*; *PressButton*" would return *true*.

The semantics of $\mathcal{A}$ is defined as follows. A state σ is a set of fluent names; we say that F holds in σ if $F \in \sigma$ and $\neg F$ hold in σ if $F \notin \sigma$. A model M of a domain description D is a pair $\langle \sigma_0, \Theta \rangle$ where σ_0 is a state and $\Theta : \textbf{actions} \times \textbf{states} \rightarrow \textbf{states}$ is a transition function such that

- for every proposition "**initially** F", the fluent literal F holds in σ_0,
- for every proposition "F **after** $A_1; \ldots; A_m$", F holds in the state $\Theta(A_m, \ldots \Theta(A_1, \sigma_0) \ldots)$,
- for every state σ, action A and fluent name F,
  - if D includes an effect proposition of the form "A **causes** F **if** $P_1, \ldots, P_n$" ("A **causes** $\neg F$ **if** $P_1, \ldots, P_n$") such that the preconditions $P_1, \ldots, P_n$ hold in σ, then $F \in \Theta(A, \sigma)$ ($F \notin \Theta(A, \sigma)$),
  - otherwise, $F \in \Theta(A, \sigma)$ iff $F \in \sigma$.

Gelfond and Lifschitz also present a mapping from $\mathcal{A}$ to a situation calculus version encoded in extended logic programming, which exploits negation-as-failure [20] as a non-monotonic operator. The mapping is sound, but it is only complete if the initial state is completely specified.[7] We will not dwell on the details of this mapping, but simply present the theory corresponding to the domain description above (the number of each line indicates what line in the original domain description it corresponds to).

$$
\begin{array}{ll}
1 & \neg Holds(credit, S0). \\
2 & \neg Holds(has\text{-}coffee, S0). \\
3a & Holds(credit, result(Load, s)). \\
3b & NonInertial(credit, Load, s). \\
4a & Holds(has\text{-}coffee, result(PressButton, s)) \\
 & \quad \leftarrow Holds(credit, s). \\
4b & NonInertial(has\text{-}coffee, PressButton, s) \\
 & \quad \leftarrow not\ \neg Holds(credit, s). \\
4c & Holds(credit, s) \leftarrow \neg Holds(has\text{-}coffee, s), \\
 & \quad Holds(has\text{-}coffee, result(PressButton, s)). \\
4d & \neg Holds(credit, s) \\
 & \quad \leftarrow \neg Holds(has\text{-}coffee, result(PressButton, s)). \\
5a & \neg Holds(credit, result(PressButton, s)) \\
5b & NonInertial(credit, result(PressButton, s))
\end{array}
\tag{3.37}
$$

[7] There are extensions of $\mathcal{A}$ that have sound and complete mappings, for instance [12].

$$
\begin{array}{ll}
Aa & Holds(f, result(a,s)) \leftarrow Holds(f,s), \\
 & \quad not\ NonInertial(f,a,s) \\
Ab & \neg Holds(f, result(a,s)) \leftarrow \neg Holds(f,s), \\
 & \quad not\ NonInertial(f,a,s) \\
Ac & Holds(f,s) \leftarrow Holds(f, result(a,s)), \\
 & \quad not\ NonInertial(f,a,s) \\
Ad & \neg Holds(f,s) \leftarrow \neg Holds(f, result(a,s)), \\
 & \quad not\ NonInertial(f,a,s)
\end{array}
$$

Note the translation of the effect propositions. 3a, 4a and 5a specify the actual effects of actions, and 3b, 4b and 5b specify when fluents are non-inert (i.e. might possibly change). In particular notice 4b, which declares *has-coffee* as non-inert with respect to *PressButton* if there is a possibility that *credit* is true (i.e. $\neg Holds(credit, s)$ cannot be proved). Aa–Ad encode a general rule of inertia: if a fluent might not possibly change (is not non-inertial), then it will have the same truth value before and after the action. Finally, 4c and 4d support postdictive reasoning: if an effect of an action appears, then the corresponding preconditions must hold, and if the effect does not appear, the preconditions must be false.

There have been a number of proposals for introducing nondeterminism and ramifications (e.g. [138]). There have also been a number of proposals for handling concurrent actions, which is investigated in more detail in chapter 4. The language $\mathcal{C}$ by Giunchiglia and Lifschitz is a related language with more general forms of propositions: static laws of the form "**caused** F **if** G" where F and G are formulas without any action symbols; and dynamic laws of the form "**caused** F **if** G **after** H" where H may contain action symbols and refers to the previous state relative to F and G. $\mathcal{C}$ can represent ramifications, nondeterministic effects, and to some extent also concurrent actions. The semantics of $\mathcal{C}$ is defined in quite a different way, though; it is based on the theory of causal explanation by McCain and Turner [93] and Lifschitz [85]. Finally, in section 3.4, we present the language $\mathcal{L}_1$ by Baral, Gelfond and Provetti [13], which is an extension of $\mathcal{A}$ that combines branching-time and narrative-based features.

Unlike SC(R), $\mathcal{A}$ is not a logic-based representation in the sense that it has a proof theory or even a syntax with logical connectives (besides a limited use of negation). Instead, the purpose of $\mathcal{A}$ is to serve as a basis for evaluation and comparison of other languages. There are representations (in particular based on logic programming) on which $\mathcal{A}$ can be mapped. These representations have the characteristics of situation calculus and therefore also the strengths and weaknesses that we discussed in connection with the

situation and fluent calculi.

3.4 The language $\mathcal{L}_1$

The language $\mathcal{L}_1$ by Baral, Gelfond and Provetti [13] is an extension of $\mathcal{A}$ [43] that lies somewhere between the branching-time formalisms presented so far and the narrative-based formalisms presented next. It has a branching event-driven temporal structure, but it also includes the possibility to refer to actual courses of action. The basic idea is that among the branches in the tree of situations, there is one partial branch starting at the initial situation and ending at the current situation that represents the course of actions that have actually occurred.[8] This actual branch can also be associated with observations. Thus, $\mathcal{L}_1$ allows an agent to keep track of the current situation and the actual course of actions and observations that lead to the current situation, and at the same time reason about hypothetical future courses of action, for instance in order to plan.

Causal laws have the same form as in $\mathcal{A}$: "A **causes** F **if** $P_1, \ldots, P_n$." The novel part of $\mathcal{L}_1$ are a number of expressions that refer to actual situations along an actual course of actions. There are fluent facts of the form "F **at** S" where S is an actual situation, occurrence facts of the forms "A **occurs_at** S" and "$[A_1, \ldots, A_n]$ **occurs_at** S", and precedence facts of the form "S_1 **precedes** S_2". These three types of expressions are collectively referred to as observations. There are two special situations: the initial situation s_0 and the current situation s_N. The following is an encoding of the coffee machine scenario in $\mathcal{L}_1$:

$$
\begin{array}{ll}
1 & \neg \textit{credit} \textbf{ at } s_0. \\
2 & \neg \textit{has-coffee} \textbf{ at } s_0. \\
3 & \textit{DrawCard} \textbf{ occurs\_at } s_0. \\
4 & \textit{PressButton} \textbf{ occurs\_at } s_1. \\
5 & s_0 \textbf{ precedes } s_1. \\
6 & \textit{DrawCard} \textbf{ causes } \textit{credit}. \\
7 & \textit{PressButton} \textbf{ causes } \textit{has-coffee} \textbf{ if } \textit{credit}. \\
8 & \textit{PressButton} \textbf{ causes } \neg \textit{credit}.
\end{array}
\tag{3.38}
$$

There is an assumption built into the semantics of $\mathcal{L}_1$ that no actions occur except for those needed to explain the observations in the particular domain description. For instance, in the domain description above, the only

[8] $\mathcal{L}_1$ is not the first language to do this. Pinto's and Reiter's situation calculus with an actual time line [111, 112] was probably the first (also see chapter 5 in this thesis).

actions that actually occur are *DrawCard* and *PressButton*, and in that order. It is also possible to infer occurrences that are not explicitly stated, as in the following domain description:

$$
\begin{array}{ll}
1 & \neg \textit{credit} \textbf{ at } s_0. \\
2 & \neg \textit{has-coffee} \textbf{ at } s_0. \\
3 & \textit{DrawCard} \textbf{ occurs\_at } s_0. \\
4 & \textit{has-coffee} \textbf{ at } s_N. \\
5 & s_0 \textbf{ precedes } s_N. \\
6 & \textit{DrawCard} \textbf{ causes } \textit{credit}. \\
7 & \textit{PressButton} \textbf{ causes } \textit{has-coffee} \textbf{ if } \textit{credit}. \\
8 & \textit{PressButton} \textbf{ causes } \neg \textit{credit}.
\end{array}
\tag{3.39}
$$

In this domain description, the explicit occurrences (that is "*DrawCard* **occurs_at** s_0") are not sufficient for explaining how *has-coffee* could be false at s_0 but true at s_N. However, if the action *PressButton* occurred immediately after *Load*, this would suffice as an explanation. In addition, this would be an explanation that would not contain any superfluous occurrences. Consequently, one can infer "[*DrawCard*, *PressButton*] **occurs_at** s_0" from the domain description above.

Besides reasoning about an actual course of actions, $\mathcal{L}_1$ also includes the possibility to reason hypothetically and even counter-factually. A hypothesis is an expression of the form "F **after** $[A_1, \ldots, A_n]$ **at** S", which is read as "the sequence $A_1, \ldots, A_n$ of actions can be executed in S, and if it were, then the fluent literal F would be true afterwards." The form "**currently** F" is an abbreviation for "F **after** [ ] **at** s_N".

The authors show how the entailment relation of $\mathcal{L}_1$ (which we do not present here) can be approximated in logic programs. The approximation is based on the assumption that the actual path of situations and occurrences is completely known. In [10], Baral, Gabaldon and Provetti provide a sound and complete characterization of the entailment relation based on circumscription.

In summary, $\mathcal{L}_1$ is a language that combines two important features of branching and narrative-based formalisms, namely hypothetical reasoning and reasoning about actual occurrences and situations. However, one important property associated with narrative-based languages that is missing in $\mathcal{L}_1$ is an independent, explicit notion of time. This is a feature of the next two formalisms, the event calculus and the Temporal Action Logics.

3.5 Event calculus

The event calculus [69] (EC) is a narrative-based representation. It has an independent notion of time, and describes actual courses of action.

3.5.1 Logic programming-based EC

In the event calculus, narratives are encoded as logic programs. In the original Event Calculus by Kowalski and Sergot [69], time was represented as both points and periods. This representation received some serious criticism [112], and later versions [70, 37, 127] have excluded periods and rely solely on time-points (integers or reals). The following presentation is based on these later versions. A more detailed discussion about EC can be found in [130].

The basic entities of EC are time-points, fluents and actions. There is a predicate $Happens(a, t)$ denoting that an action a happens at time-point t (notice that actions are instantaneous) and a predicate $HoldsAt(f, t)$ denoting that the fluent f is true at time-point t. Furthermore, there are a number of predicates relating to initialization and change of fluents: $Initially(f)$ represents that the fluent f is true at time-point 0; $Initiates(a, f, t)$ that the action a makes f true at t; and $Terminates(a, f, t)$ that the action a makes f false at t. The following is an example of an EC scenario.

$$\begin{array}{l} Happens(DrawCard, 3). \\ Happens(PressButton, 5). \\ Initiates(DrawCard, credit, t). \\ Initiates(PressButton, has\text{-}coffee, t) \leftarrow HoldsAt(credit, t). \\ Terminates(PressButton, credit, t). \end{array} \tag{3.40}$$

In addition, there are a number of rules encoding the general principle of persistence, as follows.

$$HoldsAt(f, t) \leftarrow Initially(f), not\ Clipped(0, f, t). \tag{3.41}$$

$$\begin{array}{l} HoldsAt(f, t_2) \leftarrow Happens(a, t_1), Initiates(a, f, t_1), \\ \quad t_1 < t_2, not\ Clipped(t_1, f, t_2). \end{array} \tag{3.42}$$

$$\begin{array}{l} Clipped(t_1, f, t_2) \leftarrow Happens(a, t_3), Terminates(a, f, t_3), \\ \quad t_1 < t_3, t_3 \leq t_2. \end{array} \tag{3.43}$$

Essentially, these rules state that if a fluent f is initially true ($t_1 = 0$) or made true by an action at t_1 and is not affected by any other intermediary

action, then it will remain true at a later time-point t_2. Thus, a fluent can only change when an action occurs, and the state of the world remains static between action occurrences. Note the use of negation-as-failure as non-monotonic operator in (3.41) and (3.42): if the clauses $Clipped(0, f, t)$ or $Clipped(t_1, f, t_2)$ cannot be proved, they are considered to be false and the fluent f persists. Further, due to the closed-world assumption built into logic programming, actions (*Happens*) and initiation (*Initiates*) and termination (*Terminates*) only occur where explicitly stated. Thus, there is an implicit assumption that the only actions that occur are those explicitly stated and that these actions only affect fluents when explicitly stated. The closed-world assumption also implies that the original EC can only describe one single development of the scenario. In particular, it implies that the initial state is completely determined. For instance, in the scenario in (3.40), the fact that $Initially(credit)$ is absent implies that *credit* is initially false.

3.5.2 Circumscriptive EC

There is a more sophisticated version of EC by Shanahan [129] which is based on classical logic and circumscription instead of logic programming and which permits narratives with multiple developments. The circumscriptive EC is a somewhat more complicated representation than the logic programming versions. In particular, it includes a sort for states, which are sets of (possibly negated) fluents. The reason for introducing state objects is that it permits side-stepping some minimization problems by disconnecting the effects of actions from the time-line. Thus, instead of relating *Initiates* and *Terminates* to time-points, the two predicates are associated with states. The following is an example (the action *Start* has the same function as the *Initially* predicate previously had: to set up the initial state).

$$
\begin{array}{l}
Happens(Start, 0). \\
Terminates(Start, has\text{-}coffee, s). \\
Terminates(Start, credit, s). \\
Happens(DrawCard, 3). \\
Happens(PressButton, 5). \\
Initiates(DrawCard, credit, s). \\
Initiates(PressButton, has\text{-}coffee, s) \leftarrow HoldsIn(credit, s). \\
Terminates(PressButton, credit, s).
\end{array}
\tag{3.44}
$$

In addition, there are a number of axioms that relate states to time-points (a predicate $State(t, s)$ represents the fact that the fluents in state s holds at time-point t) and that encode the general principle of persistence.

The statements about occurrences and effects of actions are assumed to completely characterize what actions occur and what fluents change. This assumption is encoded in Shanahan's circumscription policy. Shanahan minimizes *Happens*, *Initiates* and *Terminates* in parallel while varying *HoldsAt* and *State*.[9] In a later circumscriptive version of EC [130, chapter 16], Shanahan minimizes *Happens* separately from the other predicates. This enables him to eliminate the *State* predicate and state terms; consequently, this latter version is more similar to the logic-programming-based versions of EC. There are also extensions of the latter version that can represent discontinuities [103] and ramifications [131].

3.5.3 The language $\mathcal{E}$

The methodology of the language $\mathcal{A}$ has been applied to EC as well, resulting in the language $\mathcal{E}$ [58]. An $\mathcal{E}$ domain description consists of statements of the form "*A* **initiates** *F* **when** *C*" and "*A* **terminates** *F* **when** *C*", for specifying the effects of actions (*F* is a fluent name, *A* an action, and *C* is a set of fluent literals), "*A* **happens–at** *T*" for action occurrences (*T* is a time-point), and "*L* **holds–at** *T*" for observations (*L* is a fluent literal). For instance, the scenario in (3.44) can be represented in $\mathcal{E}$ as follows.

$$\begin{array}{l}
\neg \textit{has-coffee} \textbf{ holds–at } 0. \\
\neg \textit{credit} \textbf{ holds–at } 0. \\
\textit{DrawCard} \textbf{ happens–at } 3. \\
\textit{PressButton} \textbf{ happens–at } 5. \\
\textit{DrawCard} \textbf{ initiates } \textit{credit} \textbf{ when } \{\}. \\
\textit{PressButton} \textbf{ initiates } \textit{has-coffee} \textbf{ when } \{\textit{credit}\}. \\
\textit{PressButton} \textbf{ terminates } \textit{credit} \textbf{ when } \{\}.
\end{array} \tag{3.45}$$

The semantics of $\mathcal{E}$ is defined as follows. An interpretation is a mapping H : **fluent–names** $\times$ **time–points** $\rightarrow \{\textit{true}, \textit{false}\}$. Given a set of fluent literals C and a time-point T, an interpretation H satisfies C at T if for each fluent name F, if $F \in C$ then $H(F,T) = \textit{true}$, and if $\neg F \in C$ then $H(F,T) = \textit{false}$. Given a specific domain description D, a time-point T is an initiation-point (termination-point) for a fluent name F in H if for some A there is a statement "*A* **happens–at** *T*" and a statement "*A* **initiates** *F* **when** *C*" ("*A* **terminates** *F* **when** *C*" respectively) in D such that C is satisfied at T in H. Finally, a model of a domain description D is an interpretation H such that for every fluent name F and time-points T, T_1, T_2, T_3 such that

[9] There is also a predicate *AbState* that is minimized with higher priority.

$T_1 \prec T_3$ ($\prec$ is a total order on the time points), the following properties hold:

- If there is no initiation-point or termination point T_2 for F in H such that $T_1 \preceq T_2 \prec T_3$, then $H(F, T_1) = H(F, T_3)$.
- If T_1 is an initiation-point (termination-point) for F in H, and there is no termination-point (initiation-point) T_2 such that $T_1 \preceq T_2 \prec T_3$, then $H(F, T_3) = \mathit{true}$ ($H(F, T_3) = \mathit{false}$ respectively).
- For each statement of the form "L **holds–at** T" ("$\neg L$ **holds–at** T") it is the case that $H(F, T) = \mathit{true}$ (respectively $H(F, T) = \mathit{false}$).

Notice how the principle of persistence is encoded in the first two conditions. The third condition treats observations as filters on the set of models, which are not related to actual change.

Kakas and Miller have extended $\mathcal{E}$ to handle ramifications [59].

3.5.4 Discussion

The event calculus differs from SC(R) in several important respects. First, there is an independent notion of time (reals or natural numbers). Second, action occurrences are represented as statements (clauses) in EC which relate actions to time-points. This makes it possible to have temporal gaps between actions, and (at least in the circumscriptive version) to have the timing and order of actions partially specified. Likewise, observations are related to time-points instead of action sequences. Furthermore, it is straightforward to represent simultaneous action occurrences: one just has to make the temporal argument of the two actions identical.[10]

Unfortunately, the fact that action occurrences are statements, and thereby narratives are collections of statements, implies that reasoning about alternative narratives has to be done in terms of alternative logic programs/theories, as opposed to SC(R) where action sequences are terms. For instance, if one has the theory in (3.44) and one wants to reason hypothetically about what would happen if the agent did not draw the card first, then one has to:

1. Form a new theory where $\mathit{Happens}(\mathit{DrawCard}, 3)$ is not present, but the rest is as before.

[10] This concerns the fact that two actions occur at the same time. Specifying the results of concurrent actions is a different problem which can be much more difficult, as we show in chapter 4 and paper I.

2. Compute the circumscription of this theory.

3. Infer the relevant consequences from the new circumscribed theory.

Furthermore, planning (and other reasoning tasks that involve finding some choice of actions) has to be formulated meta-logically: given some action descriptions Γ_D (with initiation and termination statements), an initial state description Γ_I and a goal state description Γ_G, find a set of action occurrences Γ_A such that $\Gamma_A \wedge \Gamma_I \wedge \Gamma_D \models \Gamma_G$. Compare this to SC(R), where the same problem could be reduced to proving an existential: $\exists s[exec(s) \wedge G(s)]$.[11]

Two courses of action can be merged in EC by taking the union of the two corresponding collections of statements. For instance, one can formulate "if the actions in A lead to α, and the actions in A' lead to β given that α holds in the beginning, then A and A' together, in sequence, lead to β" as follows. If $\Gamma_A \wedge \Gamma_D \models \alpha$ and $\Gamma_{A'} \wedge \Gamma_D \models \alpha \Rightarrow \beta$ then $\Gamma_A \wedge \Gamma_{A'} \wedge \Gamma_D \models \beta$ (provided Γ_A, α, $\Gamma_{A'}$ and β are timed properly).

Reasoning about alternative developments of a specific narrative also has to be referred to on the meta-logical level. Statements of the form "these action occurrences always/necessarily have the effect α" have to be expressed meta-logically, as

$$\Gamma_I \wedge \Gamma_A \wedge \Gamma_D \models \alpha.$$

Statements of the form "these action occurrences sometimes/possibly have the effect α" have to be formulated

$$\Gamma_I \wedge \Gamma_A \wedge \Gamma_D \not\models \neg\alpha.$$

3.6 Temporal Action Logic (TAL)

The temporal action logic (TAL) is a narrative-based logic (or rather a family of logics), like EC. It originates from a logic by Sandewall [124] based on the non-monotonic policy PMON which provides a simple yet comparatively powerful solution to the frame problem. Sandewall gave a semantic characterization of PMON, and proved it correctly applicable for worlds with integer time and sequential action occurrences and with actions with extension and context-dependent and nondeterministic effects. In addition, PMON assumes complete and correct information about action laws and action occurrences

[11] Recall that computationally, a metalogical formulation need not be a disadvantage [101].

and correct information about observations. The initial observations need not completely specify the initial state, and observations can refer to states at later time-points. The timing of actions need not be completely specified.

Later, PMON was reformulated in classical logic with circumscription by Doherty and Łukaszewicz [32, 26, 27]. TAL has since been extended to represent ramifications [48], qualifications [30], concurrency (paper I, [65]) and delayed effects (paper II, [66]). Furthermore, the use of regression operators for reasoning about TAL scenarios has been investigated [16], and so has the relation to explanation closure [33]. In addition, TAL has been applied to planning [61, 60, 31]. A survey can be found in [29]. However, the presentation in this section focuses on the basic features of the original, sequential TAL, and we only briefly address ramifications.[12]

The following is the coffee machine scenario encoded in TAL. To be more specific, the scenario is encoded in the narrative description language $\mathcal{L}(ND)$ of TAL. $\mathcal{L}(ND)$ is essentially a macro-language that provides a means for compact specification of narratives. Note that the different lines are labeled. There are action laws labeled *acs*, observations labeled *obs*, and action occurrence statements labeled *occ*.

$$\begin{array}{ll} \text{acs1} & [s,t]\mathsf{Drawcard} \Rightarrow R((s,t]\mathsf{credit}) \\ \text{acs2} & [s,t]\mathsf{PressButton} \Rightarrow \\ & \quad ([s]\mathsf{credit} \Rightarrow R((s,t]\neg\mathsf{credit} \wedge \mathsf{has\text{-}coffee})) \\ \text{obs1} & [0]\neg\mathsf{credit} \wedge \neg\mathsf{has\text{-}coffee} \\ \text{occ1} & [2,5]\mathsf{DrawCard} \\ \text{occ2} & [7,8]\mathsf{PressButton} \end{array} \tag{3.46}$$

In this narrative, there are three kinds of statements. A statement of the form $[\tau,\tau']A$ represents that the action A occurs during the temporal interval $[\tau,\tau']$ (e.g. $[s,t]\mathsf{PressButton}$ or $[7,8]\mathsf{PressButton}$). A statement of the form $[\tau]\alpha$ represents that the fluent expression α holds at time-point τ (e.g. $[0]\neg\mathsf{credit} \wedge \neg\mathsf{has\text{-}coffee}$ and $[s]\mathsf{credit}$). Finally, a statement of the form $R((\tau,\tau']\alpha)$ represents that the fluent expression α becomes true somewhere in $(\tau,\tau']$ and in particular is true at τ' (e.g. $R((s,t]\mathsf{credit})$). The macro operator R stands for "reassignment", and is a tool for representing that a

[12] It should be pointed out that the TAL presented here is not identical to the one first presented by Doherty and Łukaszewicz [32, 26, 27]. The *Holds* predicate originally had only two arguments, for time and fluents. There was no type for actions, and no *Occurs* predicate. Instead, action laws were syntactic expansion schemas which were used to expand occurrence statements to precondition-effect statements. The *Occurs* predicate was introduced in [27]. Finally, reassignment was written $[s,t]f := T$ and $[s,t]f := F$ instead of $R((s,t]f)$ and $R((s,t]\neg f)$.

fluent is assigned a new value. There is an assumption that fluents can only change if they appear inside a reassignment.

Consequently, *acs1* states that if the agent draws the card through the reader then a credit is registered. We consider all free variables (s and t) to be implicitly universally quantified. *Acs2* states that if the agent presses the button and if there is a credit, then the credit is lost and a cup of coffee is obtained. Notice that the effects of DrawCard and PressButton are assumed to occur while the respective actions are executed, but we do not know exactly when. It is possible to add statements to describe what goes on during the action, though. *Obs1* is an observation of the fact that the agent has no credit and no coffee. Finally, *occ1* and *occ2* are two specific occurrences of the actions DrawCard and PressButton, respectively.

An interesting property of the R operator is that it can represent non-deterministic results. For instance, the action of flipping a coin can be represented as follows ($\otimes$ stands for exclusive or).

$$\text{acs3} \quad [s,t]\mathsf{FlipCoin} \Rightarrow R((s,t]\mathsf{heads} \otimes \mathsf{tails}) \tag{3.47}$$

This action law states that if the coin is flipped, then it will end up with either heads or tails up.

The R operator is also used in dependency constraints for specifying ramifications. The following is an example of a dependency constraint ($C_T([\tau]\alpha)$ denotes that α was false just before τ but is true at τ):

$$\text{dep1} \quad C_T([t]\mathsf{cup\text{-}broken}) \Rightarrow R([t]\neg\mathsf{has\text{-}coffee}) \tag{3.48}$$

This dependency constraint states that if cup-broken becomes true, then has-coffee becomes false.

Narratives encoded in the surface language $\mathcal{L}(ND)$ are translated to the first-order sorted language $\mathcal{L}(FL)$, which includes the predicates $Occurs : \mathcal{T} \times \mathcal{T} \times \mathcal{A}$, $Holds_i : \mathcal{T} \times \mathcal{F}_i \times \mathcal{V}_i$ and $Occlude_i : \mathcal{T} \times \mathcal{F}_i$. The subscript i indicates that there may be different fluent sorts $\mathcal{F}_i$ with different value domains $\mathcal{V}_i$, each associated with its own $Holds_i$ and $Occlude_i$ predicates. Most of the time the index will be omitted. *Occurs* represents action occurrences, *Holds* represents that fluents hold at certain time-points and *Occlude* represents that fluents can change value. The following is the translation of (3.46). Notice how reassignment (R) expressions are translated to combinations of

Holds and *Occlude*.

$$
\begin{array}{ll}
\text{acs1} & \forall s,t[Occurs(s,t,\mathsf{PressButton}) \Rightarrow \\
& \quad \forall t'[s < t' \leq t \Rightarrow Occlude(t',\mathsf{credit})] \wedge \\
& \quad Holds(t,\mathsf{credit},\mathsf{T})] \\
\text{acs2} & \forall s,t[Occurs(s,t,\mathsf{DrawCard}) \Rightarrow \\
& \quad (Holds(s,\mathsf{credit},\mathsf{T}) \Rightarrow \\
& \quad \forall t'[s < t' \leq t \Rightarrow Occlude(t',\mathsf{credit}) \wedge \\
& \quad Occlude(t',\mathsf{has\text{-}coffee})] \wedge \\
& \quad \neg Holds(t,\mathsf{credit},\mathsf{T}) \wedge Holds(t,\mathsf{has\text{-}coffee},\mathsf{T}))] \\
\text{obs1} & \neg Holds(0,\mathsf{credit},\mathsf{T}) \wedge \neg Holds(0,\mathsf{has\text{-}coffee},\mathsf{T}) \\
\text{occ1} & Occurs(2,5,\mathsf{PressButton}) \\
\text{occ2} & Occurs(7,8,\mathsf{DrawCard})
\end{array}
\tag{3.49}
$$

Given this translation, the predicates *Occurs* and $Occlude_i$ are minimized using second-order circumscription. The motivation is that only actions that have been explicitly stated to occur are assumed to occur, and only features that are explicitly affected by actions or dependencies are assumed to change. The latter assumption about no-change is encoded in the following persistence axioms in Γ_{per} (one axiom for each fluent type):

$$
\begin{array}{l}
\Gamma_{per} = \{\, \forall t, f_i, v_i[\, \neg Occlude_i(t+1, f_i) \Rightarrow \\
\qquad (Holds_i(t, f_i, v_i) \equiv Holds_i(t+1, f_i, v_i))\,]\,\}_i
\end{array}
\tag{3.50}
$$

These axioms are used for filtering the original translated narrative with $Occlude_i$ minimized, thereby effectively eliminating all models where features that are not explicitly occluded, that is to say appears within a reassignment, change.

This account for TAL has been quite brief, and we present more details in papers I and II in this thesis. Being a narrative-based formalism, TAL has many properties in common with EC: it has an explicit time line; action occurrences are represented as statements; and action occurrences and potential change are minimized globally. However, one important difference is that action occurrences have duration, and it is also possible to state what happens during the execution of an action. This makes TAL a particularly interesting candidate for representing concurrent interactions, as a realistic treatment of this subject requires that actions can overlap in time. Other interesting aspects of TAL are that it permits incomplete specifications of timing of occurrences and initial states, it can represent nondeterminstic effects, and (using occlusion) in general it supports fine-grained descriptions of what changes and what does not.

3.7 Modal logics for dynamical systems

The languages presented so far have been developed exclusively within RAC, and managing the frame problem has been a primary consideration in their development. However, there are also a number of languages for describing dynamical and reactive systems that have their origin in conventional computer science, in areas such as programming language specification and real-time systems. Here, we briefly describe two such classes of languages, partly because they have interesting properties and partly because they have been applied to RAC problems. Their interest from the perspective of this thesis lies mainly in the possibilities they offer for representing alternative developments: both dynamic logic and CTL* permit nondeterministic results in state transitions, and provide modal operators to refer to such results (as well as to different temporal relations in CTL*).

There are other approaches in RAC which are based on languages from computer science. One example is the work by Łukaszewicz and Madalinska-Bugaj [91] on reasoning about action and change using Dijkstra's semantics for programming languages [25]. The main advantage of the approach when compared to purely logical approaches is that the formula transformers (e.g. weakest precondition and strongest postcondition[13]) used in the semantics are computationally efficient.

3.7.1 Dynamic logic

Dynamic logic (DL) [116, 41] is a multi-modal logic that has been developed for describing program execution. In AI, DL has been applied to planning by Rosenschein [122], and an extension based on a preferential action semantics and intuitions from TAL has been proposed by Meyer and Doherty [100]. Like situation calculus, DL is based on branching event-driven time. The modalities in DL relate to state transitions resulting from the execution of program instructions (i.e. actions). In the context of a specific state, the DL statement $[a]\phi$ denotes that if the action a is executed, then the statement ϕ holds in all possible resulting states, and $\langle a\rangle\phi$ denotes that if a is executed, then ϕ holds in some resulting states ($\langle a\rangle\phi$ can be defined as $\neg[a]\neg\phi$). In particular, $\langle a\rangle true$ denotes that a can be successfully executed (i.e. terminates). Formally, a Kripke structure in DL is a tuple $M = \langle S, \{R_\alpha\}, L\rangle$ where S is a set of states; each $R_\alpha \subseteq S \times S$ is a transition relation for the action α; and $L : S \to \mathcal{P}(AP)$ is a function that determines what atomic propositions $p \in AP$ are true in each state $s \in S$. Given a

[13] These kinds of transformers have also been exploited in the context of TAL [16]

Kripke structure $M = \langle S, \{R_\alpha\}, L\rangle$ and a specific state $s \in S$ the truth of a statement $[a]\phi$ is defined as $M, s \models [a]\phi$ iff $M, s' \models \phi$ for all s' such that $R_a(s, s')$.

DL also contains operators for complex actions. They have the following properties.

$$\begin{array}{ll}
[a_1; a_2]\phi \equiv [a_1][a_2]\phi & \text{(sequence)} \\
[a_1 \cup a_2]\phi \equiv [a_1]\phi \wedge [a_2]\phi & \text{(choice)} \\
[a^*]\phi \Rightarrow (\phi \wedge [a][a^*]\phi) & \text{(iteration)} \\
[a^*](\phi \Rightarrow [a]\phi) \Rightarrow (\phi \Rightarrow [a^*]\phi) & \text{(iteration)} \\
[\phi?]\psi \equiv (\phi \Rightarrow \psi) & \text{(test)}
\end{array} \tag{3.51}$$

The coffee machine scenario can be encoded as follows in DL (including frame axioms).

$$[\mathit{DrawCard}]\,\mathit{credit}. \tag{3.52}$$

$$\mathit{has\text{-}coffee} \Rightarrow [\mathit{DrawCard}]\,\mathit{has\text{-}coffee}. \tag{3.53}$$

$$\neg\mathit{has\text{-}coffee} \Rightarrow [\mathit{DrawCard}]\neg\mathit{has\text{-}coffee}. \tag{3.54}$$

$$\mathit{has\text{-}card} \equiv \langle \mathit{DrawCard}\rangle \mathit{true}. \tag{3.55}$$

$$[\mathit{PressButton}]\neg\mathit{credit}. \tag{3.56}$$

$$\mathit{credit} \Rightarrow [\mathit{PressButton}]\,\mathit{has\text{-}coffee}. \tag{3.57}$$

$$\begin{array}{l}
\neg\mathit{credit} \Rightarrow (\mathit{has\text{-}coffee} \Rightarrow [\mathit{PressButton}]\,\mathit{has\text{-}coffee} \wedge \\
\quad \neg\mathit{has\text{-}coffee} \Rightarrow [\mathit{PressButton}]\neg\mathit{has\text{-}coffee}).
\end{array} \tag{3.58}$$

$$\langle \mathit{PressButton}\rangle \mathit{true}. \tag{3.59}$$

Lines (3.52–3.55) describe the *DrawCard* action, and (3.56–3.59) describe the *PressButton* action. In particular, (3.55) and (3.59) state when the two actions can be executed.

Seen as languages of RAC, there are strong similarities in the ontologies of DL and SC(R). States (possible worlds in a Kripke-style semantics) in DL correspond to situations in SC(R), and instructions/actions in DL correspond to actions in SC(R). However, there are two major differences in the representation of actions and states.

- Actions and action sequences are terms in SC(R), but modalities in DL, and therefore cannot be quantified over in the latter.
- In DL, an action can lead to zero, one or several different states, whereas in SC(R), an action always leads to a single new situation. Thus, DL permits distinguishing between necessary and possible consequences of a course of action, whereas SC(R) does not.

In paper IV of this thesis, we present a modified SC(R) that supports reasoning about alternative developments of action sequences in a manner similar to what is possible with the modal operators in DL.

3.7.2 Temporal logics

Another class of modal logics that are interesting from the perspective of RAC are temporal logics, or tense logics. The computation tree logic CTL* [21] combines features of reasoning along a time line and reasoning about alternative futures. A computation tree is a tree-shaped Kripke structure where the root is the initial state and each branch (path) represents one possible execution. A state is a set of atomic propositions (in this presentation, we restrict ourselves to the propositional case). Formally, a Kripke structure is a tuple $M = \langle S, R, L \rangle$ where S is a set of states; $R \subseteq S \times S$ is a transition relation such that each state s has at least one successor s' ($R(s, s')$); and $L : S \rightarrow \mathcal{P}(AP)$ is a function that determines what atomic propositions $p \in AP$ are true in each state $s \in S$. A path in M is an infinite sequence $s_0, s_1, \ldots$ where $s_i, s_{i+1} \in R$ for each $i \geq 0$. If $\pi = s_0, s_1, \ldots$ is a path, then the nth suffix of π is $\pi_n = s_n, s_{n+1}, \ldots$. The entailment relation $\models$ is defined both for states in a structure ($M, s \models \phi$), and for paths ($M, \pi \models \phi$).

There are two types of modal operators in CTL*. First, there are temporal operators, which concern a specific execution path π and a current state s_0 that is the start of the path.

- $\mathbf{X}\phi$ (*next*) asserts that a property ϕ holds in the next state. The semantics of $\mathbf{X}$ is defined such that $M, \pi \models \mathbf{X}\phi$ iff $M, \pi_1 \models \phi$.
- $\rho\mathbf{U}\phi$ (*until*) asserts that ϕ holds in some future state, and ρ holds in all preceding states. $M, \pi \models \rho\mathbf{U}\phi$ iff there is a k such that $M, \pi_k \models \phi$ and $M, \pi_l \models \rho$ for all $0 \leq l \leq k$.
- $\mathbf{F}\phi$ (*future, eventually*) asserts that ϕ holds in some future state. $\mathbf{F}$ can be defined in terms of the previous operators: $\mathbf{F}\phi \equiv \mathrm{T}\mathbf{U}\phi$.
- $\mathbf{G}\phi$ (*global, always*) asserts that ϕ holds in all future states. $\mathbf{G}\phi \equiv \neg\mathbf{F}\neg\phi$.
- $\rho\mathbf{R}\phi$ (*releases*) asserts that ϕ holds in all states along the path up to an including the first state where ρ holds. $\rho\mathbf{R}\phi \equiv \neg(\neg\rho\mathbf{U}\neg\phi)$.

There are also path quantifiers that describe properties of the different paths starting from a specific state s.

- $\mathbf{E}\phi$ asserts that ϕ holds in some paths. $M, s \models \mathbf{E}\phi$ iff $M, \pi \models \phi$ for some path π starting in s.
- $\mathbf{A}\phi$ asserts that a property ϕ holds in all paths starting from the current state. $\mathbf{A}$ can be defined in terms of $\mathbf{E}$: $\mathbf{A}\phi \equiv \neg\mathbf{E}\neg\phi$.

CTL* permits stating such properties as "if somebody is dead, then whatever happens he will remain dead".

$$dead \Rightarrow \mathbf{AG}dead \tag{3.60}$$

This property can also be made to hold over the entire computation tree.

$$\mathbf{AG}(dead \Rightarrow \mathbf{G}dead) \tag{3.61}$$

Actions do not have any special status in CTL*, but can be represented as atomic propositions. The following is a formalization of the effects of actions from the coffee machine scenario in CTL*, without encoding any persistence assumption.

$$\begin{array}{l}\mathbf{AG}(\textit{draw-card} \Rightarrow \mathbf{X}\textit{credit}).\\ \mathbf{AG}(\textit{press-button} \wedge \textit{credit} \Rightarrow \mathbf{X}(\neg\textit{credit} \wedge \textit{has-coffee})).\end{array} \tag{3.62}$$

As a matter of fact, the idea of occlusion from TAL can be employed in CTL* as well. Let $x\textit{credit}$ and $x\textit{has-coffee}$ represent the occlusion of *credit* and *has-coffee*. The following lines define (a) when *credit* is occluded, (b) a no-change rule for *credit*, (c) when *has-coffee* is occluded, and (c) a no-change rule for *has-coffee*, respectively.

$$\begin{array}{l}\text{a) } \mathbf{AG}((\textit{draw-card} \vee \textit{press-button}) \equiv \mathbf{X}x\textit{credit}).\\ \text{b) } \mathbf{AG}(\mathbf{X}\neg x\textit{credit} \Rightarrow (\textit{credit} \equiv \mathbf{X}\textit{credit})).\\ \text{c) } \mathbf{AG}((\textit{press-button} \wedge \textit{credit}) \equiv \mathbf{X}x\textit{has-coffee}).\\ \text{d) } \mathbf{AG}(\mathbf{X}\neg x\textit{has-coffee} \Rightarrow (\textit{has-coffee} \equiv \mathbf{X}\textit{has-coffee})).\end{array} \tag{3.63}$$

It is now possible to prove the statement that if initially there is no credit registered and the agent has no coffee, then first drawing the card and then pressing the button results in a state where the agent has a cup of coffee.

$$\begin{array}{l}\mathbf{A}(\neg\textit{credit} \wedge \neg\textit{has-coffee} \Rightarrow\\ \quad \mathbf{X}((\textit{draw-card} \wedge \neg\textit{press-button}) \Rightarrow\\ \quad\quad \mathbf{X}((\neg\textit{draw-card} \wedge \textit{press-button}) \Rightarrow \mathbf{X}\textit{has-coffee})))).\end{array} \tag{3.64}$$

CTL* is a comparatively powerful temporal logic, and there are subsets of the language that are useful. In particular, the linear temporal logic (PLTL) [115, 34] is widely used and in the context of AI has been applied to

(among other things) deductive planning [15, 8]. PLTL consists of formulas that may contain temporal operators (**X**, **F**, **G**, **U** and **R**), but not any path quantifiers (**A** and **E**). Note that if one removes the outermost **A**, then the formulae (3.62) –(3.64) are PLTL formulae. Another useful restriction (and historically a predecessor) of CTL* is CTL [36] which permits only branching-time operators — **AX** and **EX**, **AF** and **EF**, **AG** and **EG**, **AU** and **EU**, and **AR** and **ER**. Finally, there are a number of temporal logics — some of them extensions of CTL*, CTL or PLTL — that include some notion of explicit time; see [7] for an overview.

There is a parallel in temporal logics to the distinction between hypothetical developments in branching-time based RAC formalisms and actual developments in narrative-based formalisms. A theory in a branching-time temporal logic such as CTL and CTL* can be considered as representing a reactive system where each branch represents one possible trace of the system. On the other hand, a theory in a linear temporal logic such as LTL can be considered to represent one specific observed behavior of such a system [7].

An important property that distinguishes CTL* from situation calculus and dynamic logic is that no strong connection between actions and state transitions is built into CTL*. As a matter of fact, actions do not even exist as a separate type in CTL*. On the other hand, the powerful modal operators in CTL* support reasoning about alternative developments of a dynamical system in ways which are difficult to imagine in SC(R) or DL. In paper V of this thesis, we present a logic based on TAL — called the narrative logic — that has constructs for reasoning about alternative developments similar to the **A** and **E** path quantifiers in CTL*.

3.8 Comparisons

In this section, we compare the different representations in this chapter according to a number of criteria that were discussed in the previous chapter.

3.8.1 Explicit, independent notion of time

The presence of an explicit, independent notion of time facilitates expressing temporal properties of and relations between actions and states. Narrative-based representations such as EC and TAL have this property, while SC(R), $\mathcal{A}$, fluent calculus and dynamic logic do not have it. CTL* has a notion of time which is independent of actions, but it is not metric and still depends on states.

3.8.2 Action duration

Of the representations presented in this chapter, TAL is the only one that has action duration as a fundamental property. It is however possible to express action duration by introducing explicit starting and stopping events in other representations such as SC(R) and EC, although this implies that action occurrences are not primitive elements any more.

3.8.3 Alternative effects, nondeterministic effects, ramifications

All representations permit actions with context-dependent effects. Nondeterministic effects can be expressed in TAL and the fluent calculus, in some extended versions of situation calculus, $\mathcal{A}$ and the event calculus, and in dynamic logic and CTL*.

There are also extended versions of the situation calculus, $\mathcal{A}$, the fluent calculus, the event calculus and TAL that permit expressing ramifications in one way or another.

3.8.4 Reasoning about alternative courses of actions

When it comes to the ability to reason about alternative courses of actions, the representations presented in this chapter can be divided into three different groups.

Action occurrences as statements

In narrative-based formalisms such as TAL and the event calculus, action occurrences are represented with logical statements (although action types, such as *DrawCard*, are terms). Different courses of actions are represented as different sets of formulae. Thus, if one wants to compare two courses of actions (e.g. let Γ_A and Γ'_A be two closed descriptions of action occurrences), then one can do this in terms of different theories (e.g. $\Gamma_{D+I} \wedge \Gamma_A \models \alpha$ but $\Gamma_{D+I} \wedge \Gamma'_A \models \neg\alpha$, assuming Γ_{D+I} is the rest of the scenario). Another possibility is to use material implication (e.g. $\Gamma_{D+I} \models [(\Gamma_A \Rightarrow \alpha) \wedge (\Gamma'_A \Rightarrow \neg\alpha)]$.)

If one wants to quantify over courses of actions in a narrative-based formalism, then one has to do it meta-logically. Thus, if for instance one wants to say that "there exists a set of action occurrences that has the result α", then this has to be expressed as "there exists a Γ_A (with certain restrictions) such that $\Gamma_{D+I} \wedge \Gamma_A \models \alpha$".

Also in CTL*, actions would have to be represented as statements. Reasoning about alternative courses of actions can be done in terms of reasoning with formulas like (3.64).

Action occurrences as modalities

In dynamic logic, actions are modalities. Thus, it is straightforward to reason about alternative courses of action (e.g. γ and γ') simply by using these modalities (e.g. $T \models [\gamma]\alpha \wedge [\gamma']\neg\alpha$), where T is some dynamic logic theory). Notice however, that Dynamic Logic does not permit quantifying over different courses of actions; that has to be done meta-logically.

Action occurrences as terms

In SC(R), action sequences are represented as terms. Thus, it is straightforward to reason about alternative courses of actions, such as the sequences $result(a_n, \ldots result(a_1, S0)\ldots)$ and $result(a'_n, \ldots result(a'_1, S0)\ldots)$ by referring to these terms, such as

$$\begin{array}{rl} \Gamma \models & Holds(\alpha, result(a_n, \ldots result(a_1, S0)\ldots)) \wedge \\ & \neg Holds(\alpha, result(a'_n, \ldots result(a'_1, S0)\ldots)) \end{array} \tag{3.65}$$

where Γ is an SC(R) theory. Furthermore, it is possible to actually quantify over action sequences. For instance, one can formulate that "there exists a sequence of action occurrences that has the result α" simply as $\exists s[exec(s) \wedge \alpha(s)]$. The same holds for the fluent calculus.

3.8.5 Combining (merging) several courses of action

Regarding the possibility of combining different courses of actions, narrative-based formalisms such as EC and TAL have to do this by combining different sets of statements. In branching event-driven formalisms such as SC(R), the same thing can be achieved referring to different expressions representing the different courses of action. However, notice that in SC(R), action sequences are always related to a specific starting situation (typically $S0$). Consequently, one cannot refer to the action sequence $[a_1, \ldots, a_n]$ without also involving the situation where the sequence starts. In particular, there is no possibility to quantify over action sequences, and state things such as "for any action sequences s_1 and s_2 such as s_1 leads to α and s_2 leads to β provided α holds, it is the case that $s_1; s_2$ leads to β."

3.8.6 Reasoning about alternative developments of one specific course of actions

When it comes to the ability to reason about alternative developments of one specific course of actions, for instance in terms of necessary and possible results, the representations presented in this chapter can be divided into two different groups.

Alternative developments as alternative models

In all the representations based on non-modal logics, that is SC(R), $\mathcal{A}$, fluent calculus, $\mathcal{L}_1$, EC and TAL, different developments of the execution of the same set of actions will be reflected in different models. Statements of the form "these action occurrences always/necessarily have the effect α" have to be expressed meta-logically, as $\Gamma \models \alpha$. Statements of the form "these action occurrences sometimes/possibly have the effect α" have to be stated as $\Gamma \not\models \neg\alpha$. It seems paradoxical that representations such as SC(R) permit formulating logical statements about alternative courses of actions but not about different developments of the same course of action. However, as action sequences are identical to situations and each situation is only associated with one state (through the $Holds(f, s)$ predicate), this is actually the case. This is an issue which is discussed further in papers IV and V in this thesis.

Modalities over alternative developments

In contrast to the non-modal logics discussed above, both dynamic logic and CTL* permit formulating logical statements about different developments of the same course of action. In dynamic logic, the statement "these action occurrences always/necessarily have the effect α" can be expressed as $[\gamma]\alpha$, and "these action occurrences sometimes/possibly have the effect α" can be expressed as $\langle\gamma\rangle\alpha$. If we use the encoding of action occurrences presented in section 3.7.2, the corresponding expressions in CTL* are of the forms $\mathbf{A}(\mathbf{X}(a_1 \Rightarrow \mathbf{X}(a_2 \Rightarrow \ldots \mathbf{X}\alpha \ldots)))$ and $\mathbf{E}(\mathbf{X}(a_1 \wedge \mathbf{X}(a_2 \wedge \ldots \mathbf{X}\alpha \ldots)))$.

3.9 Summary

In this chapter, we have examined a number of important representations used within (and to some extent also outside) RAC. The table in figure 3.1 summarizes the results. In the next two chapters, we present some

Criteria	SC(R)	$\mathcal{A}$, $\mathcal{L}_1$,	FL	EC	TAL	DL	CTL*
Logic	Classical	Special, LP	Classical, LP	LP, Classical	Classical	Modal	Modal
Explicit time	No	No	No	Yes	Yes	No	No (but ext.)
Action duration	No	No	No	No	Yes	No	No
Nondeterministic effects	Yes (ext.)	Yes (ext.)	Yes	Yes	Yes	Yes	Yes
Ramifications	Yes (ext.)	Yes (ext.)	Yes	Yes	Yes	Yes	Yes
Action occurrences	Terms	NA	Terms	Statements	Statements	Modalities	Statements
Quant. over action occ.	Yes	NA	Yes	No	No	No	No
Alt. dev. for same actions	Models	Models	Models	Models	Models	Modalities	Modalities

Table 3.1: Summary of properties. *Ext* means that there are extensions that have the property in question, and NA means that the property is not applicable.

representations for concurrent actions and interactions, and some attempts to combine features of branching event-driven time and explicit, independent time.

Chapter 4

Concurrent interaction

In chapter 3, we presented a number of formalisms for reasoning about action and change, but one property that was not addressed was the ability to describe how simultaneously occurring actions might interact with each other. As a matter of fact, there have been few attempts to systematically address the problem of concurrent interactions in RAC. In this chapter, we present a selection of particularly interesting proposals which are based on different principles for dealing with interactions. due to Georgeff [45], Pinto [112, 113], Baral and Gelfond [11], Bornscheuer and Thielscher [17, 17] and Pelavin [109].

A more complete but less detailed survey appears at the end of paper I, covering the following approaches:

- Hendrix's work [51] on representing continuous and simultaneous processes.
- Lansky's GEM [77] which focuses on structural relations between events.
- Thielscher's theory of dynamic systems [135], which includes concepts such as natural state transitions and momentary fluents.
- Ferber and Müller's theory for dynamic multi-agent environments [38] which has a distinction between influences and state.
- A number of approaches in situation calculus by Gelfond, Lifschitz and Rabinov [44], Lin and Shoham [90] and Reiter [120].

4.1 Georgeff

Georgeff's situation calculus [45] represents one of the earliest (1987) attempts to formalize concurrent actions. It is a formalism with some significant differences from more traditional situation calculus versions. In particular, there is a sort $\mathcal{W}$ for world histories, which are sequences of world states (situations in traditional situation calculus terminology). Furthermore, there is no *result* function. Instead, the function $succ(s, w)$ denotes the successor world state of a world state s in a history w, and $occurs(e, s)$ denotes that the event (or action) e occurs in the world state s. Thus, it is possible for an action to have different results in different world histories, even when executed in the same world state. On the other hand, there are no terms representing action sequences (i.e. situation terms in traditional situation calculus). Finally, the predicate $holds(\theta, s)$ denotes that the relational feature θ holds in s.

In order to determine which features are not affected by an event, Georgeff introduces a predicate $indep(p, e, s)$ to represent that feature p is *independent* of event e in state s. Thus, by providing axioms of the form

$$holds(\gamma, s) \wedge occurs(e, s) \Rightarrow holds(\theta, succ(s, w)) \tag{4.1}$$

and

$$holds(\delta, s) \Rightarrow indep(\theta, e, s) \tag{4.2}$$

one can specify what changes and what does not change due to an event. A persistence axiom

$$\begin{aligned} \forall w, s, p[\ & holds(p, s) \wedge \neg holds(p, succ(s, w)) \Rightarrow \\ & \exists e[occurs(e, s) \wedge \neg indep(p, e, s)]] \end{aligned} \tag{4.3}$$

states that a feature can change only due to an event which the feature depends on. Note that there might be several events occurring in the same world state. If a feature changes, then it must depend on at least one of these events.

Georgeff then introduces the concept of *correctness condition* with the predicate $cc(p, e, s)$. This represents the fact that any event that does not interfere with condition p will not interfere with (prevent) the event e. Thus, if the condition

$$\begin{aligned} \exists \phi, \psi[\ & cc(\phi, e_1, s) \wedge cc(\psi, e_2, s) \wedge \\ & indep(\phi, e_2, s) \wedge indep(\psi, e_1, s)\ ] \end{aligned} \tag{4.4}$$

holds, then the two events e_1, e_2 can occur in the same world state s. There is also a notion of causality, in the form of axioms

$$\begin{aligned}\forall w, s, \phi, e_1, e_2[\ & (causes_s(\phi, e_1, e_2) \wedge holds(\phi, s) \wedge \\ & occurs(e_1, s)) \Rightarrow occurs(e_2, s)] \\ \forall w, s, \phi, e_1, e_2[\ & (causes_n(\phi, e_1, e_2) \wedge holds(\phi, s) \wedge \\ & occurs(e_1, s)) \Rightarrow occurs(e_2, succ(s, w))].\end{aligned} \tag{4.5}$$

Thus, causal relations between events can be stated in terms of the $causes_s$ and $causes_n$ predicates. The causal laws play a role similar to the dependency laws in TAL. However, causality is defined for pairs of events, and this limits it usefulness for modeling concurrent interactions. For instance, Lifschitz' and colleagues' soup bowl example [44], which is used in the subsequent sections to illustrate a number of other approaches to concurrency, seems difficult to encode in Georgeff's SC. The soup bowl example involves two events (lifting the left side of the bowl and lifting the right side) that in different combinations can generate a third event (the bowl is entirely lifted or the soup is spilled). This cannot be expressed with binary relationships between events.

Georgeff then goes on to address processes, which essentially are related groups of events with limited interaction with events outside the process. The point of representing processes is to explicitly restrict the number of possible interactions between events, and thereby reduce combinatorial complexity. Fluents and events can be internal to a process P (denoted with $internal_f(\phi, P, s)$ and $internal_e(e, P, s)$), or they can be external ($external_f(\phi, P, s)$ and $external_e(e, P, s)$). The correctness conditions of internal events may only depend on internal fluents:

$$\begin{aligned}\forall e, s, P\ [\ & internal_e(e, P, s) \Rightarrow \\ & \exists \phi[internal_f(\phi, P, s) \wedge cc(\phi, e, s)]\ ].\end{aligned} \tag{4.6}$$

Similar constraints are imposed on preconditions of internal events (Georgeff does not provide any details). Furthermore, internal fluents are always independent of non-internal events.

$$\begin{aligned}\forall e, s, \phi, P\ [\ & internal_f(\phi, P, s) \wedge \neg internal_e(e, P, s) \Rightarrow \\ & indep(\phi, e, s)].\end{aligned} \tag{4.7}$$

Besides the internal and external events of a given process, there can also be events that belong to neither category. These events (called *ports*) permit limited interaction between internal and external events. Georgeff also provides a number of compositional operators from concurrency theory [53]: prefixing, sequencing, non-deterministic choice, and parallelism.

In summary, Georgeff's formalism can define when two events (actions) can and cannot occur simultaneously, and what the result is when two independent events are executed simultaneously. However, in its current form, the formalism is restricted to binary relationships between events; this is why we did not manage to encode the soup-bowl example. A limitation is the lack of an explicit notion of duration, so events cannot partially overlap. The causal laws produce events, and are thereby inherently integrated into the framework for concurrency.

4.2 Pinto

In order to introduce concurrent actions in the situation calculus, the first thing one has to do is to decide how to represent concurrent actions as terms, as is the case with single actions. In his Ph.D. thesis [112], Pinto extends SC[1] with a function $+ : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ such that $a_1 + a_2$ denotes the action of performing a_1 and a_2 concurrently. There is also the predicate $\in: \mathcal{A} \times \mathcal{A}$ such that $a_1 \in a_2$ denotes that a_1 is part of the concurrent action a_2, and a predicate $primitive : \mathcal{A}$ defining those actions that are not further decomposable.

Pinto identifies two different problems with concurrent actions, namely the precondition interaction problem and the effect interaction problem. In short, the precondition interaction problem concerns how to obtain $Poss(a_1 + a_2, s)$ i.e. the preconditions for executing a_1 and a_2 concurrently, provided that one has adequate definitions of $Poss(a_1, s)$ and $Poss(a_2, s)$. One example is an agent G that is in a situation where he can paint two different walls W_1 and W_2 in two different colors C_1 and C_2: $Poss(paint(G, W_1, C_1)) \wedge Poss(paint(G, W_2, C_2))$. Surely, we do not want to infer that the agent is capable of performing these two actions concurrently: $Poss(paint(G, W_1, C_1) + paint(G, W_2, C_2))$. This would imply a simultaneous capacity of agent G which is unrealistic by human standards. In order to represent the assertion that two actions have interacting preconditions, Pinto suggests using a predicate $precInt : \mathcal{A} \times \mathcal{A}$. Two actions are then defined to be executable concurrently if they can be executed in isolation and their preconditions do not interact:

$$Poss(a_1 + a_2, s) \equiv \\ (Poss(a_1, s) \wedge Poss(a_2, s) \wedge \neg precInt(a_1, a_2)). \tag{4.8}$$

[1] Pinto's extension is based on Reiter's SC, but has some small differences. For instance, Pinto uses reification of fluents.

Of course, it remains to axiomatize *precInt*, and this turns out to be highly domain-dependent. Pinto gives one example based on the use of resources. Let $xres(a, r)$ denote that the action a requires the exclusive use of the resource r and let $sres(a, r)$ denote that a requires the use of the sharable resource r. For instance, we can declare the following resource requirements for the painting action.

$$\begin{aligned} &xres(paint(g, w, c), g) \wedge xres(paint(g, w, c), w) \wedge \\ &xres(paint(g, w, c), c) \wedge sres(paint(g, w, c), Light) \end{aligned} \tag{4.9}$$

Given information on what resources different actions require, one can then determine what actions have interacting preconditions by comparing their resource requirements:

$$\begin{aligned} &precInf(a_1, a_2) \equiv (a_1 \notin a_2 \wedge a_2 \notin a_1 \wedge \\ &\quad \exists r[(xres(a_1, r) \wedge xres(a_2, r)) \vee (sres(a_1, r) \wedge xres(a_2, r)) \vee \\ &\qquad (xres(a_1, r) \wedge sres(a_2, r))]). \end{aligned} \tag{4.10}$$

Recall that in SC, the result of an non-executable action is totally undefined; such an action can result in any situation. This also applies to concurrent actions. As a matter of fact, it is sufficient to have two actions with interacting preconditions in a concurrent action in order to make the latter non-executable, independently of how many other actions it consists of. Thus, precondition interactions have a global effect on a concurrent action, and are not restricted to the subactions that actually are incompatible. For instance, if $paint(G, W_1, C_1) + paint(G, W_2, C_2)$ is not executable, then neither is $paint(G, W_1, C_1) + paint(G, W_2, C_2) + sing(F) + dance(H) + open(I, Door1)$. Although this approach might seem a bit coarse, it makes sense if one considers $\neg Poss(c, s)$ to represent the fact that c cannot occur (e.g. it is "inconsistent" to state that c occurs in s). Compare this to the interpretation of $\neg Poss(c, s)$ as "a will fail"; in this case, it would be more intuitive to assume that the other actions can succeed although the two painting actions fail.

The effect interactions problem concerns how to determine the effects of the concurrent execution of a pair of actions, given the effects of the individual actions. Pinto identifies two kinds of effect interaction, namely cancellation and synergy, and proposes to model them as ramifications of the individual actions. One example is Pinto's encoding of Lifschitz' and colleagues' soup bowl example [44]. There are two actions *LiftLeft* and *LiftRight* for lifting the left and the right side of the bowl respectively. There are also the fluents $lifted_l$ and $lifted_r$ representing that the soup bowl is lifted

on either side, *lifted* representing that the bowl is actually lifted from the table, and *spilled*. There are the following effect axioms:

$$Poss(c,s) \wedge LiftLeft \in c \Rightarrow Holds(lifted_l, s)) \quad (4.11)$$
$$Poss(c,s) \wedge LiftRight \in c \Rightarrow Holds(lifted_r, s))$$

Two domain constraints determine what happens when the bowl is lifted on both or on just one side ($\otimes$ is exclusive or).

$$Holds(lifted, s) \equiv (Holds(lifted_l, s) \wedge Holds(lifted_r, s)) \quad (4.12)$$
$$Holds(spilled, s) \equiv (Holds(lifted_l, s) \otimes Holds(lifted_r, s))$$

Consequently, lifting the bowl on the left or right side respectively has the direct effect of having the bowl lifted on that side, and having the bowl totally lifted or the soup spilled are indirect effects.

In a recent paper [113], Pinto proposes an approach to concurrent interactions which has some similarity to the approach presented in paper I. The idea is to use natural events (in the style of Reiter [120]) to model causal dependencies, and then to use causal dependencies to model concurrent interactions. For instance, the soup bowl scenario is modeled using two natural actions *Spills* and *Nspills* that cause spill to become true and false respectively. Being natural actions, they occur whenever they can occur.

$$\forall a, s[Poss(\{a\}, s) \wedge natural(a) \Rightarrow occurs(a, s)] \quad (4.13)$$

The soup bowl scenario is encoded as follows. First, there are the agent-invoked actions *Liftleft* and *LiftRight*.

$$Poss(\{LiftLeft\}, s) \Rightarrow \neg Holds(lifted_l, s) \quad (4.14)$$
$$Poss(\{LiftRight\}, s) \Rightarrow \neg Holds(lifted_r, s) \quad (4.15)$$
$$Poss(c,s) \wedge LiftLeft \in c \Rightarrow Holds(lifted_l, do(c, a)) \quad (4.16)$$
$$Poss(c,s) \wedge LiftRight \in c \Rightarrow Holds(lifted_r, do(c, s)) \quad (4.17)$$

The concurrent interactions between these two actions are then encoded in terms of the natural actions *Spills* and *Nspills*, which start and end the process of spilling respectively. Line (4.18) declares these two actions to be natural, lines (4.19) to (4.22) declare when they occur (i.e. whenever the antecedents hold) and lines (4.23) to (4.24) declare their effects.

$$natural(Spills) \wedge natural(Nspills). \quad (4.18)$$
$$\neg Holds(lifted_l, s) \wedge Holds(lifted_r, s) \wedge \neg Holds(spilling, s) \Rightarrow \quad (4.19)$$
$$Poss(\{Spills\}, s).$$

$$Holds(lifted_l, s) \wedge \neg Holds(lifted_r, s) \wedge \neg Holds(spilling, s) \Rightarrow Poss(\{Spills\}, s). \quad (4.20)$$

$$Holds(lifted_l, s) \wedge Holds(lifted_r, s) \wedge Holds(spilling, s) \Rightarrow Poss(\{Nspills\}, s). \quad (4.21)$$

$$\neg Holds(lifted_l, s) \wedge \neg Holds(lifted_r, s) \wedge Holds(spilling, s) \Rightarrow Poss(\{Nspills\}, s). \quad (4.22)$$

$$Poss(c, s) \wedge Spills \in c \Rightarrow Holds(spilling, do(c, s)). \quad (4.23)$$

$$Poss(c, s) \wedge Nspills \in c \Rightarrow \neg Holds(spilling, do(c, s)). \quad (4.24)$$

Notice that one dependency is expressed in at least two axioms: one precondition axiom for defining the conditions that trigger some change (via a natural action) and one that defines the nature of the change (through the same natural action). For instance, (4.19) and (4.23) define a causal dependency from lifting on just the right side to spilling the soup. The same dependency, that is lifting on the right side but not the left side causes the soup to be spilled, could have been expressed in TAL as follows:

$$C_T([t]\neg lifted_l \wedge lifted_r) \Rightarrow R([t]spilled) \quad (4.25)$$

In paper I, we provide a more elaborate version of the soup bowl example encoded in TAL-C, an extension of TAL. There, we notice some similarity between the use of *influences* in TAL-C and the use of natural actions like *Spills* and *Nspills* above.[2] What is particularly interesting is that Pinto motivates his approach with arguments for modularity; using causal dependencies to model concurrent interactions on the level of fluents allows us to describe actions in isolation.

In summary, Pinto addresses a fairly wide range of problems associated with concurrency, including precondition interactions due to for instance resource constraints, and cases of synergy and cancellation of effects. It is especially interesting to see the use of domain constraints and natural events for representing effect interactions. Yet, one major limitation is the fact that actions have no explicit duration. Thus, it is not possible to represent different ways for actions to temporally overlap, which is something actions tend to do in many realistic domains.

[2] Yet, we should point to two important differences between Pinto's natural actions and the influences in TAL-C. First, influences can have temporal extension. Second, Pinto does not take advantage of natural actions in order to resolve conflicts or combine effects. His presentation is restricted to the soup bowl example.

4.3 Concurrency in $\mathcal{A}_C$ and $\mathcal{A}^+{}_C$

With their $\mathcal{A}_C$ language [11], Baral and Gelfond take a different approach. Instead of deriving combined effects of actions from their individual effects, they suggest directly defining the effects of concurrent actions.

$\mathcal{A}_C$ is an extension of Gelfond's and Lifschitz's $\mathcal{A}$ language (see chapter 3). The only syntactical difference is that action names are nonempty finite sets of primitive actions. For instance, the following is a domain description in $\mathcal{A}_C$ encoding the soup bowl scenario, where *lift_l* and *lift_r* are the actions of lifting the bowl on the left and right side respectively.

initially $\neg$*spilled*. (4.26)

{*lift_l*} **causes** *spilled*. (4.27)

{*lift_r*}**causes***spilled*. (4.28)

{*lift_l*, *lift_r*} **causes** $\neg$*spilled* **if** $\neg$*spilled*. (4.29)

Line (4.26) is a v-proposition (value proposition) that states that the soup is not spilled in the initial state, and the rest of the lines are e-propositions (effect propositions) that state the different effects of lifting the bowl. The last e-proposition (4.29) is required in order to override the effects of the two previous ones.

The semantics of concurrent actions in $\mathcal{A}_C$ is based on three principles. First, effects are inherited from smaller to larger action names. For instance, the e-proposition

{*lift_l*} **causes** *spilled*

implies

{*lift_l*, *whistle*} **causes** *spilled*

(provided *whistle* does not cause $\neg$*spilled*). Second, if there are several e-propositions with conflicting effects on the same fluent, then larger action names have precedence over smaller ones. Thus, line (4.29), in virtue of its action name being a superset of those of the two lines (4.27) and (4.28), has precedence over the latter. Third, if there are several applicable e-propositions with conflicting effects that are not ordered according to a subset relation, the outcome is undefined.

Formally, matters are arranged as follows. We say that the execution of an action a in a state σ *immediately causes* a fluent literal f to be true if there is an e-proposition "a **causes** f **if** $p_1, \ldots, p_n$" such that $p_1, \ldots, p_n$ hold in σ. Further, the execution of an action a in a state σ *causes* a fluent literal f to be true if

1. a immediately causes f in σ, or

2. a inherits the effect f from its subset in σ, i.e. there is a $b \subset a$, such that execution of b immediately causes f in σ, and there is no c such that $b \subset c \subseteq a$ such that c immediately causes $\neg f$ in σ.

Next, we define the following sets representing the positive and negative effects of an action a in a state σ:

$$E^+(a,\sigma) = \{f \mid f \text{ is a fluent name and execution of } a \text{ in } \sigma \text{ causes } f\},$$

$$E^-(a,\sigma) = \{f \mid f \text{ is a fluent name and execution of } a \text{ in } \sigma \text{ causes } \neg f\}.$$

Now, we can define the concept of a model in $\mathcal{A}_C$. A structure (σ_o, Φ) is a model of a domain description D if:

1. Every v-proposition from D is true in (σ_o, Φ);

2. For every action $a = \{a_1, \ldots, a_n\}$ and every state σ,

   (a) if $E^+(a,\sigma) \cap E^-(a,\sigma) = \emptyset$, then $\Phi(a,\sigma)$ is defined and $\Phi(a,\sigma) = \sigma \cup E^+(a,\sigma) \setminus E^-(a,\sigma)$.

   (b) otherwise, $\Phi(a,\sigma)$ is undefined.

For instance, if $\sigma = \emptyset$, then $E^+(\{\mathit{lift\_l}, \mathit{lift\_r}\}, \sigma) = \emptyset$; although the two e-propositions (4.27) and (4.28) for separately lifting on the left and right sides are applicable, the last e-proposition has precedence over both of them. On the other hand, $E^-(\{\mathit{lift\_l}, \mathit{lift\_r}\}, \sigma) = \{\mathit{spilled}\}$. Consequently, $\Phi(\{\mathit{lift\_l}, \mathit{lift\_r}\}, \sigma) = \emptyset \cup \emptyset \setminus \{\mathit{spilled}\} = \emptyset$.

Point (2b) above implies that whenever there is a pair of subactions with contradictory effects in a concurrent action, the entire next state is undefined. Some authors, such as Bornscheuer and Thielscher [17, 18], have pointed out that that is in some cases too strong an assumption.[3] Bornscheuer and Thielscher consider the following example, which involves a dog

[3] In the case of the precondition interaction problem (see section 4.2), it could make sense to view a concurrent action that contained some subactions with conflicting preconditions as "impossible". However, in the $\mathcal{A}_C$ example above, it is the *effects* that are in conflict. In this case, it would be more reasonable to assume that (some of) the conflicting actions fail than to assume that the actions are impossible to (attempt to) execute concurrently.

sleeping beside a door with an electric opener. If the door opener is activated, the dog will wake up. You can close the door by pulling it, and you can open the door by running into it and at the same time activating the electric door opener. However, just running into the door will hurt you.

initially *sleeps* (4.30)
$\{activate\}$ **causes** $\neg sleeps$
$\{run\_into\}$ **causes** *hurt* **if** $\neg open$
$\{pull\}$ **causes** $\neg open$
$\{activate, run\_into\}$ **causes** *open*
$\{activate, run\_into\}$ **causes** $\neg hurt$ **if** $\neg hurt$

If we consider the concurrent action $a = \{activate, pull, run\_into\}$, we observe that given the semantics of $\mathcal{A}_C$, and assuming that the initial state $\sigma_0 = \{sleeps\}$, we have that $E^+(a, \sigma_0) = \{open\}$ and $E^-(a, \sigma_0) = \{sleeps, hurt, open\}$. As $E^+(a, \sigma_0) \cap E^-(a, \sigma_0) = \{open\}$, the next state $\Phi(a, \sigma_0)$ is undefined. Therefore, one can not even show that the dog is no longer asleep ($\neg sleeps$), although this actually did not have anything to do with the conflict between *activate* and *pull*.

Bornscheuer and Thielscher suggest keeping the conflict local, and just letting it affect the very fluent that caused it, that is *open*. Their suggestion is realized in a language called $\mathcal{A}_C^+$ [17], which is an extension of $\mathcal{A}_C$, and which treats conflicting effects of concurrent actions as cases of implicit nondeterminism. In a later paper, they present another language $\mathcal{A}_{NCC}$ [18] which in addition to concurrent actions can represent explicit nondeterminism.[4] They also present translations from $\mathcal{A}_C^+$ respectively $\mathcal{A}_{NCC}$ to the fluent calculus. The presentation to follow is based on $\mathcal{A}_C^+$, but concurrency is treated in essentially the same way in $\mathcal{A}_{NCC}$. We continue to use the notation from $\mathcal{A}_C$.

The idea of Bornscheuer and Thielscher is that if the intersection F of the sets of positive and negative effects of an action a in a state σ_0 is nonempty, $F = E^+(a, \sigma_0) \cap E^-(a, \sigma_0) \neq \emptyset$, then the values of the fluents in F are indeterminate, but all other fluents behave normally. For instance, in the scenario above, $\neg sleeps$ and $\neg hurt$ hold after a, but *open* can either hold or not hold. Technically, this is achieved by modifying the semantics of $\mathcal{A}_C$ as follows.

First, Φ is not a function but a ternary relation that takes a state, an action name and another state as arguments. Informally, $\Phi(\sigma, a, \sigma')$ represents

[4] Explicit nondeterminism means that one can explicitly state that an action or action combination can have alternative effects. $\mathcal{A}_{NCC}$ includes a construct a **alternativelycauses** $e_1, \ldots, e_n$ **if** $c_1, \ldots, c_n$ for this purpose.

that doing a in σ can possibly result in σ'. Second, by using a transition relation, an action sequence can correspond to multiple state sequences. Therefore, it becomes necessary to be able to identify which one of these state sequences is intended when v-propositions (of the form f **after** $a_1, \ldots, a_n$) are evaluated. For this purpose, a function φ is introduced. $\varphi([a_1, \ldots, a_n])$ denotes one of the states corresponding to $a_1, \ldots, a_n$ in which a v-proposition f **after** $a_1, \ldots, a_n$ would be evaluated.

Now, a structure $(\sigma_0, \Phi, \varphi)$ is a model of a domain description D if:

1. $\varphi([]) = \sigma_0$,

2. $(\varphi([a_1, \ldots, a_{m-1}]), a_m, \varphi([a_1, \ldots, a_m]) \in \Phi$,

3. $(\sigma, a_m, \sigma') \in \Phi$ iff $\sigma' = (\sigma \cup E^+(a, \sigma) \setminus E^-(a, \sigma)) \cup E^*$ for some $E* \subseteq (E^+(a, \sigma) \cup E^-(a, \sigma))$, and

4. for all v-propositions f **after** $a_1, \ldots, a_n$ in D, f holds in $\varphi([a_1, \ldots, a_n])$.

Given the semantics of $\mathcal{A}_C^+$, we have that if $\sigma_0 = \{sleeps\}$ and $a = \{activate, pull, run\_into\}$, then (σ_0, a, σ') iff $\sigma' = \emptyset$ or $\sigma' = \{open\}$. That is to say, $\neg sleeps$ and $\neg hurt$ hold after a, but *open* is indeterminate.

In $\mathcal{A}_C$ and $\mathcal{A}_C^+$, concurrent interactions are modeled on the level of actions (with the exception of conflicting effects in $\mathcal{A}_C^+$). The fact that effect propositions with larger action names override those with smaller contributes to elaboration tolerance [95]. Yet, $\mathcal{A}_C$ and $\mathcal{A}_C^+$ do not permit describing actions in isolation, and therefore cannot be considered to be particularly modular. As a matter of fact, if two domain descriptions were to be merged, one would have to go through all new action combinations in order to identify potential interactions. This is quite opposite to the approaches of Karlsson and Gustafsson (see paper I) and Pinto [113] (see previous section), who argue for describing actions separately and instead modeling interactions on the level of fluents by using causal dependencies. $\mathcal{A}_C^{[+]}$ is restricted to propositional domains with instantaneous actions and it lacks any notion of ramifications. As interactions are modeled on the level of actions, extending $\mathcal{A}_C^{[+]}$ to also include ramifications — which of course can interact with actions and other ramifications but which are not directly associated with any actions — appears to be a very difficult problem. In general, it is questionable whether it is possible to extend $\mathcal{A}_C^{[+]}$ to more complex ontologies.

4.4 Pelavin

In [109], Pelavin presents a sophisticated language for reasoning about plans with concurrent actions and external events. The language is based on Allen's interval logic [6], where time is represented as intervals. These intervals can be related in different ways; for instance, two intervals i_1 and i_2 can meet, that is i_1 ends where i_2 starts: $\mathit{Meets}(i_1, i_2)$, one can be contained inside the other: $\mathit{In}(i_1, i_2)$, or one can be before (without meeting) the other $\mathit{Before}(i_1, i_2)$.

4.4.1 Semantics

The semantics of Pelavin's logic is based on the notion of a world history. A world history captures exactly what the planning agent does and what happens in the external world. All world histories have a common time line. Given a specific world history, the properties and relations that hold during different intervals and the actions and events that occur are represented as propositions. For instance, $\langle i_1, h\rangle \in \mathit{has\text{-}coffee}$ means that the property *has-coffee* holds during the interval i_1 in the context of world history h. Two world histories can have parts that are identical. In particular, the accessibility relation $R(i, h1, h2)$ represents that two world histories $h1$ and $h2$ share a common past through the end of the interval i. Thus, the R relation arranges world histories into a tree structure, where two world histories are on the same branch as long as R connects them.

To capture the effects of basic actions, Pelavin introduces a closeness function CL. $CL(ba, i, h)$ represents the set of world histories $\{h'_k\}_k$ resulting from executing the basic action ba during the interval i in the world history h. An old world history h and a new one h'_k should differ only on account of the basic action ba in the interval i. For instance, if h is a world history with no action occurrences at all such that $\langle i, h\rangle \notin \mathit{credit}$ holds for any i (that is all the time), then $CL(\mathit{DrawCard}, I_1, h)$ would result in a singleton set $\{h\}$ of new world histories such that $\langle I_1, h'\rangle \in \mathit{DrawCard}$ holds and in addition $\langle i, h\rangle \in \mathit{credit}$ holds for any i such that $\mathit{Meets}(I_1, i)$.

Each action is associated with some standard conditions (similar to the *Poss* predicate in SC) that determine when the action is executable. If the standard conditions do not hold for an action a at i, then $CL(ba, i, h) = \{h\}$, that is, the closeness function returns an unaltered world history.

A plan instance is essentially a set of action occurrences (action-interval pairs). The effects of a plan instance are computed by applying the CL function recursively. Thereby, the semantics of Pelavin's logic can capture

such things as when two concurrent action attempts interfere with each other (and perhaps only one or none of the actions actually occurs), when one action negates the standard condition of another action (prevention), and when one action brings about the standard condition of another action (enablement).

4.4.2 Syntax

The syntax of Pelavin's logic includes terms for temporal intervals and plan instances; $DrawCard(I_1)$ is one example, denoting the primitive action *DrawCard* with interval I_1. There are temporal predicates for instance *has-coffee*(i); eternal (time-invariant) predicates such as $Meets(i_1, i_2)$ and $Occ(pi)$; the standard logical connectives and quantifiers; and the two modal operators $INEV$ and $IFTRIED$. The $Occ(pi)$ predicate represents that the plan instance pi actually occurs.[5] The inevitability operator $INEV$ is used to describe the branching time structure, and its semantics is defined in terms of the R relation. The statement $INEV(i, s)$ means that the statement s holds in all branches that are possible at the end of time i. For instance,

$$INEV(I_1, \forall i_2[Meets(I_1, i_2) \Rightarrow dead(i_2)])$$

means that some person is inevitably dead after I_1. The $IFTRIED$ operator captures how individual plan instances (sets of action occurrences) affect the world, and its semantics is defined in terms of the CL relation. $IFTRIED(pi, s)$ means that if the plan instance pi is attempted, then s holds in all resulting world histories. In particular, executability of a plan instance can be defined as follows.

$$Executable(pi) =_{def} IFTRIED(pi, Occ(pi))$$

The following is an example of the use of the $IFTRIED$ operator, which concludes what happens if the button of the coffee machine is pressed.

$$\begin{array}{l} \forall i_1, i_2[\ credit(i_1) \wedge Meets(i_1, i_2) \Rightarrow \\ \qquad IFTRIED(PressButton(i_2), \\ \qquad\quad \exists i_3[Meets(i_2, i_3) \wedge has\text{-}coffee(i_3) \wedge \neg credit(i_3)]\ )\ ] \end{array}$$

[5] *Occ* is considered an eternal (time-invariant) predicate as it does not have any temporal argument, although the plan instances it takes as arguments contain temporal information.

4.4.3 Domain descriptions

For specifying the executability conditions and effects of actions, the $INEV$ operator is used. The general form for specifying an executability condition is as below, where pi is a plan instance – often just a single action – and EC is the condition.

$$INEV(Ip, EC \Rightarrow Executable(pi))$$

For instance, the following is an executability condition for the $DrawCard$ action.

$$INEV(Ip, \textit{has-card}(i_1) \wedge Meets(i_1, i_2) \Rightarrow \\ Executable(DrawCard(i_2)))$$

The general form for specifying effects is as follows, where pi is a plan instance, and EFF are the effects.

$$INEV(Ip, Occ(pi) \Rightarrow EFF)$$

For instance, the following specifies the effects of the $DrawCard$ action.

$$INEV(Ip,\ Occ(PressButton(i_1)) \Rightarrow \\ \exists i_2[Meets(i_1, i_2) \wedge \textit{credit}(i_2)])$$

Conditions and effects need not be restricted to the very start and end of actions. As a matter of fact, Pelavin's logic is very expressive when it comes to temporal relations. For instance, one can state that the effect of sailing across a lake (from point a to point b) is that the boat is moving while the sailing takes place.

$$INEV(Ip, Occ(Sail(a, b, i)) \Rightarrow Moving(i))$$

4.4.4 Frame axioms

Pelavin only briefly sketches how the frame problem can be tackled in his logic. He suggests using frame axioms of the general form

$$INEV(Ip, C \Rightarrow (P \Rightarrow IFTRIED(pi, P)))$$

where for a given property P and plan instance pi the formula C are the conditions under which the frame axiom applies. In short, a frame axiom states that "under the condition C, if P holds, then P would still hold if ip was executed." For instance,

$$INEV(Ip, \exists i0[\neg \textit{credit}(i_0) \wedge Meets(i_0, i)] \Rightarrow \\ (\neg \textit{has-coffee}(i) \Rightarrow IFTRIED(PressButton(i), \neg \textit{has-coffee}(i)))).$$

Note that these frame axioms, like those in section 2.5.1, chapter 3, have to associate no-change with specific actions. Therefore, it is unlikely that Pelavin's logic can handle any more complex ramifications.

4.4.5 Simultaneous effects

Pelavin's logic permits quite a sophisticated treatment of concurrency. In particular, the $IFTRIED$ operator makes it possible to describe actions in isolation. It allows us to describe the effects of an action as a difference between the case when the action is performed and the case when it is not. For instance, assume there is a box of coins.[6] The predicate $CoinsIn(b, n, i)$ represents the fact that there are n coins in the box b during the time interval i, and the plan instance $TakeCoin(a, b, i)$ represents that agent a takes one coin out of the box b. One can model the effect of taking a coin from the box in terms of the difference between the number of coins there are in the box if the action $TakeCoin(a, b, i)$ is not executed, and the number of coins (one less) there would be if the action was executed.

$$\begin{array}{l} INEV(Ip, \forall a, b, n[\neg Occ(TakeCoin(a, b, i)) \wedge \\ \quad \exists i_2, [Finishes(i_2, i) \wedge CoinsIn(b, n, i_2) \wedge n \geq 1] \Rightarrow \\ \quad IFTRIED(TakeCoin(a, b, i), \\ \qquad \exists i_3[Finishes(i_3, i) \wedge CoinsIn(b, n-1, i_3)])]) \end{array} \tag{4.31}$$

Pelavin presents an example involving two agents – the planning agent and another agent $agt2$ – which both occasionally use a terminal. There is an action $UseTerm(i)$ for the planning agent, and there is a predicate $UsingTerm(agt2, i)$ meaning that the other agent is using the terminal. Only one of the agents can use the terminal at a time, as captured in the following constraint.

$$\begin{array}{l} \forall i_1, i_2[\neg Disjoint(i_1, i_2) \Rightarrow \\ \quad INEV(Ip, \neg(UsingTerm(agt2, i_1) \wedge Occ(UseTerm(i_2))))] \end{array} \tag{4.32}$$

Pelavin discusses three types of interaction possible in this scenario. The first case is when the other agent always has priority over the planning agent. The planning agent cannot use the terminal if the other agent is using it, but he can himself be interrupted by the other agent. This case can be encoded as (4.32) and the following specification.

$$\begin{array}{l} \forall i, pi[INEV(Ip, UsingTerm(agt2, i) \Rightarrow \\ \qquad\qquad IFTRIED(Ip, UsingTerm(agt2, i_1)))] \end{array} \tag{4.33}$$

[6] This is an example from paper I. Pelavin presents a similar example involving the buying and paying for a CD record.

The second case is the reverse, when the planning agent has priority. This case is captured in (4.32) and the following specification, which asserts that $UseTerm$ is executable under all circumstances.

$$\forall i[INEV(Ip, Executable(UseTerm(i)))] \tag{4.34}$$

The third case is when the agent that first tries to use the terminal gets it. If both agents try at the same time, then the planning agent gets it. This priority relationship can be formulated as the planning agent being able to use the terminal if and only if the other agent is not using the terminal during a non-disjoint time interval that starts previously.

$$\begin{aligned}\forall i[\, & INEV(Ip, \neg\exists i0[(Overlaps(i0, i) \vee Finishes(i, i0) \vee \\ & Contains(i0, i)) \wedge UsingTerm(agt2, i0)] \equiv \\ & Executable(UseTerm(i)))]\end{aligned} \tag{4.35}$$

In general, Pelavin's logic is able to capture a fairly wide range of phenomena associated with concurrency. Besides the terminal example, Pelavin also presents examples involving interaction due to resource constraints, and combinations of forces acting on an object. A limitation is that all treatment of concurrent interactions must involve references to actions (due to the use of the $IFTRIED$ operator), which implies that treating interactions due to ramifications is apparently not possible.

4.4.6 Conclusion

Pelavin offers one of the most sophisticated treatments of concurrency in the literature, and indeed also of the representation of plans.

- His logic is capable of representing concurrent interactions in flexible ways.
- It permits describing actions in isolation.
- It is the only treatment of concurrent interactions in the context of actions with duration that we are aware of, besides paper I in this thesis.
- The modal constructs can be used for reasoning about alternative courses of actions (plan instances in Pelavin's vocabulary), as well as about alternative developments of a given course of action. The $IFTRIES$ operator has some similarities to the modal constructs for necessary and possible consequences in the narrative logic (NL) presented in paper V in this thesis, and the $INEV$ operator resembles the NL general law operator.

Besides these strengths, the logic also has a number of drawbacks.

- The logic has a highly complex, non-standard semantics.
- The logic relies on frame axioms.
- Representing ramifications seems very difficult to do due to the reliance on frame axioms.

4.5 Concurrency in computer science

In computer science, the study of concurrent systems is based on the notion of a process — a possibly non-terminating behavior pattern. The smallest components of a process are events (or actions or instructions). Being designed entities, processes in computer systems typically have well-defined boundaries and inner structure. Yet, the facts that processes execute concurrently and that the order of events between different processes is at best partially specified introduce an element of nondeterminism which makes concurrent systems considerably more difficult to design and test than strictly sequential ones. In addition, for for instance an embedded system, the environment (which can be considered a process as well) contains elements of nondeterminism. The central aim of the field of concurrency in computer science is to develop principles for designing and verifying concurrent systems. In particular, one is interested in liveness properties (something good eventually happens) and safety properties (something bad never happens) over the different runs of the system (i.e. the different ways the system can evolve.)

To give a flavor of the field, we present a fragment of the process algebra CCS [104] (Calculus of Communicating Systems) for specifying concurrent systems, and we discuss some modal logics for describing properties of such systems. The intention is to give a foundation for comparisons between concurrency in RAC and in computer science. We should also mention that concepts and languages from concurrency theory have had some influence on RAC. For instance, Georgeff [45] and Lansky [77] make use of operators from concurrency theory in order to structure actions into processes, and so do De Giacomo and colleagues [24] in their concurrent robot programming language ConGolog. Chen and De Giacomo [19] use an extension of CCS to model concurrent interactions between actions in the sense discussed in this chapter, which is exemplified with the soup bowl example.

4.5.1 Calculus of Communicating Systems

CCS [104] permits describing the behaviors of processes in a compositional way; complex processes can be described in terms of combinations of simpler ones. The following is a very simple process — a clock that perpetually ticks — described in CCS:

$$Cl \stackrel{def}{=} \mathsf{tick}.Cl \tag{4.36}$$

The prefix operator (.) is defined as

$$a.E \stackrel{a}{\rightarrow} E$$

which means that the process $a.E$ may perform the action a and evolve into the process E. In the inference rules of CCS, the notation $E \stackrel{a}{\rightarrow} F$ occurs frequently, representing the fact that a process E may become F by performing an action a. The following inference rule defines the $\stackrel{def}{=}$ relation (the premises are on the top, and the consequent is in the lower part),

$$\frac{E \stackrel{a}{\rightarrow} F}{P \stackrel{a}{\rightarrow} F} \quad P \stackrel{def}{=} E$$

If $P \stackrel{def}{=} E$ and E can evolve into F, then P can evolve into F. In the clock example, Cl evolves into $\mathsf{tick}.Cl$, which in turn, when performing tick, evolves into Cl again, and so on.

The choice operator + is defined by the following inference rules,

$$\frac{E_1 \stackrel{a}{\rightarrow} F}{E_1 + E_2 \stackrel{a}{\rightarrow} F} \quad \frac{E_2 \stackrel{a}{\rightarrow} F}{E_1 + E_2 \stackrel{a}{\rightarrow} F}$$

If the process E_1 (or E_2) may become F, then $E_1 + E_2$ may also become F.

The following is an example of a vending machine. One can either insert 2p, choose a big chocolate bar and collect it, or one can insert 1p, select a small chocolate bar and collect it.

$$\begin{aligned} V &\stackrel{def}{=} \mathsf{2p}.V_b + \mathsf{1p}.V_l \\ V_b &\stackrel{def}{=} \mathsf{big}.V_c \\ V_l &\stackrel{def}{=} \mathsf{little}.V_c \\ V_c &\stackrel{def}{=} \mathsf{collect}.V \end{aligned} \tag{4.37}$$

The concurrency operator | can be used to compose processes that are to be executed concurrently. Concurrent processes can evolve without communication, according to the first two of the following inference rules, or with communication according to the third.

$$\frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F} \qquad \frac{F \xrightarrow{a} F'}{E|F \xrightarrow{a} E|F'} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{\overline{a}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

In short, the concurrent composition $E|F$ may evolve by evolving either of the constituents. In the case of communication, if E may input on channel a and evolve into E' and F may output on a and evolve into F' (or vice versa) then $E|F$ may evolve into $E'|F'$. Here, a and $\overline{a}$ are co-actions (e.g. input and output of a value on a channel) that cannot be performed in isolation; one presupposes the other.

There are a number of other operators, for instance a conditional form and a form for action restrictions, that are not addressed here.

4.5.2 Modal logics for concurrent processes

The capabilities of processes can be described logically. Hennessy-Milner logic [52] is a very simple modal logic that has two kinds if modal operators: $[a]\Phi$ describing a process such that if it starts with a then it must evolve into Φ; and $\langle a\rangle\Phi$ which if starting with a might evolve into Φ.

If a process E has the property Φ, we write $E \models \Phi$. The satisfaction relation $\models$ can be defined as follows.

$$E \models \mathtt{tt} \tag{4.38}$$

$$E \not\models \mathtt{ff} \tag{4.39}$$

$$E \models \Phi \wedge \Psi \quad \text{iff} \quad E \models \Phi \text{ and } E \models \Psi \tag{4.40}$$

$$E \models \Phi \vee \Psi \quad \text{iff} \quad E \models \Phi \text{ or } E \models \Psi \tag{4.41}$$

$$E \models [a]\Phi \quad \text{iff} \quad \forall F \in \{E' \mid E \xrightarrow{a} E'\}.F \models \Phi \tag{4.42}$$

$$E \models \langle a\rangle\Phi \quad \text{iff} \quad \exists F \in \{E' \mid E \xrightarrow{a} E'\}.F \models \Phi \tag{4.43}$$

For instance, one can express that the vending machine process has the property that one can insert 1p and press small and collect, but one cannot insert 2p and select small.

$$V \models \langle \mathsf{1p}\rangle\langle \mathsf{small}\rangle\langle \mathsf{collect}\rangle \mathsf{tt} \wedge [\mathsf{2p}][\mathsf{small}]\mathsf{ff} \tag{4.44}$$

Hennessy-Milner logic is only capable of expressing local properties; it cannot express properties that hold for instance over the entire run of a

system. In order to express such properties, one needs logics with more powerful temporal operators, such as CTL* [21] which was presented in chapter 3. An alternative is to introduce extremal fix-point operators as in the temporal mu-calculus [35]. The modal mu-calculus [72] combines the type of modal operators used in Hennessy-Milner logic with fix-point operators.

4.5.3 Discussion

The notion of concurrency in computer science is quite different from the notion of concurrency in the RAC approaches presented in the previous sections of this chapter. The difference is due to the difference between the notion of a process in computer science and the notion of an action in RAC. A process is an evolving, often nonterminating, entity — it is primarily the process itself that is subject to change, and how it changes may be affected by how other processes around it evolve. On the other hand, an action occurrence is typically a time-bound entity that is not changing in itself, but primarily causes change in the states of the environment in which it is executed. The nature of that change may be influenced by other, simultaneous action occurrences. Often, as in SC(R), the action occurrence is just a transition from one state to the next. Of course, the difference between a process "changing in itself" and an action occurrence "changing the state of the environment" should not be exaggerated — a process may have a state that includes variables, and which might evolve with the process as a whole.

For the representation of actions, there might yet be something interesting to learn from process algebra. As long as action occurrences cause transitions from one state to the next, the notion of an "evolving" action does not make any sense. If action occurrences have temporal extension, however, as is the case in TAL, and can be affected by other simultaneous action occurrences, considering actions as evolving entities could be a fruitful approach. This is not realized in the concurrent extension to TAL presented in paper I in this thesis; there, the state may evolve differently due to concurrent interaction but the action occurrence itself has no time-dependent properties. In the work of Chen and De Giacomo [19], which is based on an extension of CCS, actions do not have temporal extension. Consequently, the idea of evolving occurrences has to be left for future work. Due to the designed nature of processes in computer science, something like CCS might not be sufficient, though; processes in nature are perhaps not as well-behaved and well-delimited as those in computers.

Processes need not be identified with actions — they can also be seen as a means for structuring actions into composite behaviors. That is what, for instance, Georgeff [45], Lansky [77] and De Giacomo and colleagues [24] (in ConGolog) do, and this is primarily also the approach of Chen and De Giacomo [19].

4.6 Summary

In this chapter, we have given in-depth presentations of a number of approaches to concurrent actions. They all used different methods of dealing with interactions.

Georgeff's approach is based on deriving interactions from the properties of events. There is an explicit representation of independence: when a fluent that is independent of an event is not affected by that event. Likewise, a correctness condition determines what fluents can prevent a specific event.

Pinto relies on deriving precondition interactions indirectly: he defines a relation that determines when the preconditions of two actions interfere in terms of properties (such as resources used) of the individual actions. In addition, Pinto uses domain constraints and (in a later paper) natural events to indirectly define the effects of concurrent actions. This can be viewed as representing a second, somewhat different approach, where the treatment of interactions need not directly involve any references to the actions that caused them.

A third approach was proposed by Baral and Gelfond, and extended by Bornscheuer and Thielscher. Instead of dealing with interactions indirectly, via the fluents that the actions affected and depended on, they dealt with interactions directly, using rules that referred to sets of co-occurring actions. By having more specific rules overriding less specific ones, a certain degree of elaboration tolerance was achieved. However, their approach makes it impossible to describe actions in isolation, and thus it suffers from a lack of modularity.

All the previous work is based on an SC type of ontology, and therefore lack an explicit notion of action duration. Pelavin's logic, on the other hand, is based on Allen's interval logic, where there is an independent notion of time. Pelavin's logic can be considered to represent a fourth approach to concurrent interactions. Using the $IFTRIES$ modality, Pelavin is able to specify concurrent interactions in terms of the difference between what the result would be if an action was executed and what the result would if it was not executed.

In short, we have seen the following ways to represent concurrent interaction.

- Indirectly via properties of actions.
- Via indirect effects (encoded as domain constraints or natural actions).
- Directly in action laws.
- In terms of the difference between executing and not executing an action.

Of these different approaches to handling interactions, all but the third one support describing actions in isolation. Only the second approach appears to have the capacity to deal with interactions that involve indirect effects. All the others require explicit references to the actions involved in the interaction. Obviously, an indirect effect does not have any explicit references to the actions that caused it.

We also covered concurrency as it is viewed in computer science, and we have in particular looked at Milner's CCS [104] and some modal logics for concurrent systems. From an RAC perspective, we have observed that concurrent processes in computer science could be used to represent individual action occurrences and how they evolve — an approach that has not yet been investigated — or to structure actions into behaviors.

In paper I, we address the problem of concurrent interactions in the context of actions with duration. We present an approach that is based on encoding concurrent interactions in terms of indirect effects, and which can describe interactions not only between actions, but also between ramifications and even delayed effects and processes. (Delayed effects and processes are addressed in paper II.) Furthermore, we attempt to systematically address a wide range of interactions. We consider the issue of modularity; our language TAL-C permits describing actions and causal dependencies in isolation.

Chapter 5

Enriching the representation of narratives

Recall the discussion in chapter 3 about the relative strengths and weaknesses of narrative-based formalisms such as TAL and event-based branching formalisms such as the situation calculus (SC). The former supported rich temporal properties of and relations between actions and fluents but had weak support for hypothetical reasoning, whereas the latter made hypothetical reasoning easy but had a very restricted notion of time. In this chapter, we analyze a number of attempts by Pinto [112, 113] and Reiter [120] to combine the features of narrative-based languages and SC. These attempts all use the situation calculus as a starting point, and although they manage to incorporate notions of explicit time and other features of narrative-based formalisms, they are still subject to limitations inherent to the situation calculus.

Actually, both Reiter [120] and Pinto [113] address a wider problem than just adding a time-line to situation calculus. Reiter is interested in continuous change and natural events, and Pinto [113] in causal relations between action occurrences.

Finally, there have also been proposals that start from a narrative-based language and then add branching-time features. We briefly address two such approaches, by Sandewall [124] and Provetti [117].

5.1 Adding a time-line to situation calculus

Some of the early attempts to introduce narrative-like features into SC involved a concept of an actual path of situations. Some examples are work

by Miller and Shanahan [102] and Pinto [112], and some attempts to combine the situation calculus and the event calculus [71, 14]. Here, we will concentrate on early work by Pinto, as presented in his PhD thesis [112].

5.1.1 Time line

Pinto introduces a notion of explicit time (i.e. non-negative reals) into SC by associating situations with starting and ending times. Situations are periods of no-change that start and end at specific times, and actions are instantaneous transitions between situations. The relations between time, situations and actions are axiomatized as follows.

$$\begin{array}{l} \forall a, s[end(s, a) = start(result(a, s))]. \\ \forall a, s[start(s) < start(result(a, s))]. \\ start(S0) = 0. \end{array} \tag{5.1}$$

Note that the *end* function takes two arguments: one situation argument and one action argument. This implies that the ending time of a situation depends on what action ends the situation. Thus, given a specific situation, different alternative actions can end the situation at different time-points. Strangely enough, it is not possible to choose among alternative time-points for the same action. For instance, given the initial state, one does not have the possibility to choose between executing the $DrawCard$ action at 3.5 and at 6.4, as the statements $end(S0, DrawCard) = 3.5$ and $end(S0, Drawcard) = 6.4$, which represent these two choices, are mutually inconsistent.

5.1.2 Actual path

In order to represent narrative-style action occurrences, Pinto proposes using an "actual" path of situations. The basic idea is that among the different paths in a tree of situations, there is one distinguished path, the actual path, that is considered to represent the course of events actually realized, whereas other paths represent hypothetical courses of actions that could have happened. A predicate $actual : \mathcal{S}$ is introduced for specifying what situations lie along the actual path. The axioms for *actual* are as follows.

$$\begin{array}{l} actual(S0). \\ \forall a, s[actual(result(a, s)) \Rightarrow actual(s) \wedge Poss(a, s)]. \\ \forall a, a', s[actual(result(a, s)) \wedge actual(result(a', s)) \Rightarrow a = a'] \end{array} \tag{5.2}$$

These axioms state that the initial situation is actual, and that the actual situations must form a single unbroken path. Notice that there are no restrictions on whether the path is finite (i.e. has an actual situation with no actual successor) or infinite.

5.1.3 Actual action occurrences

Next, Pinto introduces a predicate $occurs : \mathcal{A}$ to represent actual action occurrences, i.e. occurrences along the actual path of situations. The predicate is defined as follows.

$$\forall a, s[occurs(a, s) \equiv actual(result(a, s))] \tag{5.3}$$

Further, there is a predicate $occurs_{\mathcal{T}} : \mathcal{A} \times \mathcal{T}$, that relates action occurrences to time-points (compare to $Occurs : \mathcal{T} \times \mathcal{T} \times \mathcal{A}$ in TAL and $Happens : \mathcal{A} \times \mathcal{T}$ in EC), a predicate $holds_{\mathcal{T}} : \mathcal{F} \times \mathcal{T}$ that relates fluents to time-points (compare to $Holds : \mathcal{T} \times \mathcal{F} \times \mathcal{V}$ in TAL and $HoldsAt : \mathcal{F} \times \mathcal{T}$ in EC) and a predicate $during : \mathcal{T} \times \mathcal{S}$ that relates time-points to actual situations:

$$\begin{array}{l} \forall a, t[occurs_{\mathcal{T}}(a, t) \equiv \exists s[occurs(a, s) \wedge start(result(a, s)) = t]\,] \\ \forall f, t[holds_{\mathcal{T}}(f, t) \equiv \exists s[actual(s) \wedge during(t, s) \wedge holds(f, s)] \\ \forall t, s[during(t, s) \equiv \\ \quad (actual(s) \wedge start(s) < t \wedge \\ \quad\ \forall a[occurs(a, s) \Rightarrow end(s, a) \geq t]\,)\,] \end{array} \tag{5.4}$$

Thereby, it is possible to encode narrative-style information. For instance, the following is an encoding of the coffee machine scenario with time-points.

$$Poss(DrawCard, s) \equiv Holds(has\text{-}card, s). \tag{5.5}$$

$$Poss(PressButton, s). \tag{5.6}$$

$$Poss(DrawCard, s) \Rightarrow Holds(credit, result(DrawCard, s)). \tag{5.7}$$

$$\begin{array}{l} Poss(PressButton, s) \wedge Holds(credit, s) \Rightarrow \\ \quad (Holds(has\text{-}coffee, result(PressButton, s)) \wedge \\ \quad\ \neg Holds(credit, result(PressButton, s))). \end{array} \tag{5.8}$$

$$\begin{array}{l} \exists s_1, s_2[s_1 = result(DrawCard, S0) \wedge start(s_1) = 2.4 \wedge \\ \quad s_2 = result(PressButton, s_1) \wedge start(s_2) = 5.0 \wedge actual(s_2)] \end{array} \tag{5.9}$$

The novel part of this axiomatization is axiom (5.9), which selects an actual (partial) path and sets the time of the situations and actions along this path. Pinto also presents a circumscriptive policy for minimizing $Occurs_{\mathcal{T}}$, which

makes it possible to directly state what occurs. For instance, axiom (5.9) could be replaced with the following.

$$Occurs_{\mathcal{T}}(DrawCard, 2.4) \wedge Occurs_{\mathcal{T}}(PressButton, 5.0). \tag{5.10}$$

Given (5.10), minimizing $Occurs_{\mathcal{T}}$ would result in the two situations $s_1 = result(DrawCard, S0)$ and $s_2 = result(PressButton, s_1)$ being the only actual situations.

5.1.4 Conclusion

In conclusion, Pinto's extension to the situation calculus manages to incorporate features of both SC and narrative-based logics such as TAL and EC. Where it falls short, however, is in integrating these features in a manner that actually permits representing facts that are beyond the abilities of both standard SC and narrative-based formalisms. In particular, Pinto's representation of narratives suffers from the same limitations as pure narrative-based languages: narratives are collections of logical statements, and therefore one cannot reason deductively about alternative narratives and quantify over narratives. On the other hand, the non-actual branches (which are outside the narrative) are not accessible to narrative-style reasoning, in particular reasoning about explicit time. Thus, it is questionable if anything really is gained using Pinto's proposal. This is a shortcoming that his proposal shares with several other proposals which attempt to combine narrative-style and SC-style features [102, 117].

5.2 Temporal, concurrent situation calculus

The shortcomings of the type of approaches discussed in the previous section have to a considerable extent been overcome in later work by Reiter [120] and Pinto [114]. In this section, we present the work of Reiter. Pinto's work is presented in section 5.3.

5.2.1 Actions with a temporal argument

Recall the problems mentioned in section 5.1 about associating situations with start and end times, and of encoding narrative-style information using an actual path. A seemingly trivial but important remedy is to introduce a temporal argument in action names [120]. This makes it possible to define the timing of an action as follows.

$$time(\mathbf{A}(\overline{x}, t)) = t \tag{5.11}$$

For instance, the action occurrences in the timed coffee machine scenario can be represented with the term[1]

$$result(\{PressButton(5.0)\}, result(\{DrawCard(2.4)\}, S0)). \tag{5.12}$$

Notice that this approach presupposes that the timing (or at least the ordering) of actions is completely specified. This is an inherent property of the use of the *result*-function in SC, which imposes a total temporal ordering on the situations and actions along a specific path. Thus, it is not possible to write for example

$$result(\{PressButton(T1)\}, result(\{DrawCard(T2)\}, S0)) \tag{5.13}$$

and then add a constraint

$$T1 < T2 \tag{5.14}$$

as $PressButton(T1)$ has to occur after $DrawCard(T2)$ on the path in question.

5.2.2 Actions with duration

It can be argued that a theory that presupposes that all actions are instantaneous is hardly suitable for representing many realistic domains, e.g. physical domains, where the execution of an action takes time. As suggested by Pinto [112] and Ternovskaia [134], action duration can be represent by conceiving actions as processes that are started and ended by instantaneous actions. For instance, a *pickup* action can be represented by the fluent $picking\_up(x)$ and the two actions $start\_pickup(x,t)$ and $end\_pickup(x,t)$. A consequence of this solution is that actions are no longer primitive concepts of the language. Furthermore, as the timing of the starting and ending actions need to be completely specified, it presupposes that the durations of all actions are completely known (and, it seems, always the same, which excludes context-dependent durations).

5.2.3 Simultaneous action occurrences

Besides action duration, Reiter incorporates another feature into his temporal situation calculus that is straightforward in narrative-based languages but more difficult in SC: simultaneous action occurrences. Following a proposal by Lin and Shoham [90] among others, Reiter represents concurrent

[1] The brackets around the actions are due to the treatment of simultaneous action occurrences, as explained below.

actions as sets of simple actions. The *result*-function is modified to take concurrent actions as arguments; this is the reason that the actions in the term representing coffee machine scenario (5.12) are inside set brackets:

$$result(\{PressButton(5.0)\}, result(\{PressButton(2.4)\}, S0)). \tag{5.15}$$

The notation $a \in c$ is used to represent the fact that the simple action a is one of the actions constituting the concurrent action c.

Concurrent actions are required to be coherent, that is they must contain at least one simple action, and all simple actions must occur at the same time.

$$\begin{array}{l} \forall c[coherent(c) \equiv \\ \quad \exists a[a \in c] \wedge \exists t \forall a'[a' \in c \Rightarrow time(a') = t]\,] \end{array} \tag{5.16}$$

The *time* function can be extended to concurrent actions, as follows.

$$\begin{array}{l} \forall c, t, s[coherent(c) \Rightarrow \\ \qquad [time(c) = t \equiv \exists a[a \in c \wedge time(a) = t]\,] \wedge \\ \qquad start(result(c, s)) = time(c)\,] \end{array} \tag{5.17}$$

Similarly, the *Poss* predicate can be extended to concurrent actions. The following conditions should hold for $Poss : \mathcal{C} \times \mathcal{S}$.

$$\begin{array}{l} \forall a, s[Poss(a, s) \Rightarrow Poss(\{a\}, s)] \\ \forall c, s[Poss(a, s) \Rightarrow (coherent(c) \wedge \forall a[a \in c \Rightarrow Poss(a, s)]\,] \end{array} \tag{5.18}$$

5.2.4 Natural actions and continuous change

Reiter elaborates Pinto's work [112] on natural actions and continuous change. Natural actions are actions that occur spontaneously (independently of any agent) when certain conditions hold. The *Poss* predicate is used to specify the conditions for when natural actions occur. The occurrence of natural actions is made possible by the fact that properties of the world are not necessarily static during the time-span of a situation, but can be subject to continuous change. One example due to Reiter is a falling ball that bounces when it reaches the floor. This example is encoded as follows.

$$\begin{array}{l} Poss(bounce(t), s) \equiv (\ is\_falling(s) \wedge \\ \quad [\ height(s) + vel(s) \cdot (t - start(s)) - 1/2g(t - start(s))^2 = 0\,]\ ) \end{array} \tag{5.19}$$

The functions *height*(s) and *vel*(s) represent the height above the floor and the velocity of the ball at the beginning of s. The axiom states that bouncing is possible if the ball has fallen the entire distance to the floor since the start

of the situation. If bouncing becomes possible, then the current situation ends with the occurrence of a bounce (the result of the bouncing action is not described).

The axiom above is not without problems. Note that the description of how the ball falls is encoded in the precondition for the bouncing action. It seems it would have been more convenient to provide a separate description of the falling of the ball, and then simply state that *bounce* is possible when the height above the floor is 0 and the ball is moving downward. In particular, such an approach would provide a higher degree of modularity; as it is now, the formula for a falling ball has to be repeated in the conditions for each natural action (e.g. bounce on floor, crash through glass pane, splash into water) that can be triggered by a falling ball. In addition, describing the falling of the ball in the preconditions of *bounce* does not support queries about the height of the ball above the floor at different time-points. There is simply no way to ask e.g. "what is $height(s)$ at time-point 5.4", because *height* does not take temporal parameters.

A situation term in the temporal, concurrent situation calculus consists of both agent-initiated actions and natural actions. In standard situation calculus, some situation terms represent non-executable actions sequences; namely those where the preconditions of some action (as specified by the *Poss* predicate) are not satisfied. By the introduction of natural actions, situation terms can also be impossible due to the fact that a natural action that should have occurred did not occur. One example is a situation where an agent drops the ball and it does not bounce when it reaches the floor. Reiter introduces the concept of a legal situation to distinguish those situations that represent a possible course of events from those that do not. We will not go into the details of how this is done; in short, a legal situation is one where (a) all actions are executable, and (b) whenever a natural action can occur, it does occur. An important concept in the context of legal situations is the least natural time-point of a situation. It denotes the time-point where the first (natural or agent-initiated) action occurs, and if the situation is legal, then it also signifies the end-time of the situation.

5.2.5 Conclusion

Reiter's approach is the first to actually incorporate narrative features into the branching time-structure of SC.[2] It also supports representing natural events and continuous change, and although it does not appear to surpass

[2] As observed in the previous section, earlier works only permitted narrative-style information in one (actual) path. Reiter permits narrative-style information in all paths.

earlier work on that topic in a narrative-based context (see for instance Sandewall [123] and Shanahan [128]), to incorporate such features into SC still has to be considered a significant accomplishment.

Yet, the approach has a number of shortcomings. In particular, the timing of each action and natural event must be determinate. Thus, Reiter's SC does not possess a temporal expressiveness equivalent to a narrative-based formalism such as TAL, where the timing and even order among actions can be partially specified. The limitations of the approach also become evident in later work by Reiter [121], where he combines the temporal, concurrent situation calculus with the agent programming language GOLOG [80]. Here, the narrative is actually executed by an agent embedded in a real world. As the actual execution of the narrative is subject to all the disturbances and uncertainty of the real world, there will obviously always be temporal discrepancies between the previously defined narrative and the actual execution. Using a temporally more expressive narrative-based language, the problem of the mismatch between the time-stamps in the narrative and the actual times could be solved simply by leaving the timing of actions partially specified in the narrative, and then fix the timing when actions actually occur.

Another problematic issue is that all natural actions have to appear in the situation terms. If one specifies a situation solely in terms of the actions the reasoning agent intends to perform, then that situation might be illegal, as the natural actions are missing. It would have been beneficial to separate the description of what the agent does, and what are the consequences of his doings.

In general, Reiter's SC is based on very strong assumptions about completeness. Whether a specific situation is legal depends on the precise timing and ordering of actions (both natural and executed by the agent) as well as the initial state. If the initial state is partially specified, or if the timing of actions is not exactly known, then the same sequence of actions on the agent's behalf can generate different natural actions and consequently different situation terms in different models.

5.3 Occurrences and narratives as constraints in the branching structure

In later work, Pinto [113] presents another elaboration of SC that permits expressing narrative-style information. As in the work of Reiter, it is based on a concept of legal situations. However, Pinto provides a richer notion of

legality, which supports reasoning about for instance occurrences at specific times ("the sun will rise tomorrow at 6:03"), triggered occurrences that can involve some delay ("if you eat the forbidden fruit, you will be expelled"), preventable occurrences ("the train to Ottawa leaves every day at 7 p.m., provided there is not a strike or snow storm or ...") and even partially specified occurrences (e.g. "it will eventually stop raining (but we don't know when)."

5.3.1 Actions, time and histories

As in [112], Pinto uses a function $start : \mathcal{S} \rightarrow \mathcal{T}$ to denote the starting time of a situation,[3] and concurrent actions are represented as described in the previous section. An important conceptual development is that situations are considered to have a history. Therefore, there are constructs for referring to what has happened in the history of a situation. There is a predicate $hasOccurred : (\mathcal{A} \cup \mathcal{C}) \times \mathcal{T} \times \mathcal{S}$ representing that in the history that lead to a situation, an action occurred at a specific time-point, like in $hasOccurred(DrawCard, 5, result(PressButton, result(DrawCard, S0)))$. This predicate is defined as follows for concurrent and single actions.

$$\begin{array}{l} hasOccurred(c, t, s) \equiv \\ \quad \exists s'[result(c, s') \preceq s \wedge start(do(c, s')) = t] \end{array} \tag{5.20}$$

$$hasOccurred(a, t, s) \equiv \exists c[a \in c \wedge hasOccurred(c, t, s)] \tag{5.21}$$

There is also a predicate $Sit : \mathcal{T} \times \mathcal{S} \times \mathcal{S}$, which represents that given the history of a situation, a specific time-point is included in a specific preceding situation along that history.

$$\begin{array}{l} Sit(t, s, s_h) \equiv \\ \quad \exists c[result(c, s) \preceq s_h \wedge start(s) < t \wedge t \leq start(result(c, s))] \end{array} \tag{5.22}$$

For instance, if it is the case that $start(result(DrawCard, S0)) = 5$ and further that $start(result(PressButton, result(DrawCard, S0))) = 7$, then time-point 6 is included in the former situation:

$$\begin{array}{l} Sit(6, result(DrawCard, S0), \\ \quad result(PressButton, result(DrawCard, S0))). \end{array}$$

[3] Strangely, there are still no temporal arguments in actions, which again prevents us from reasoning about alternative narratives where the same actions occur but at different times.

5.3.2 Action occurrences as global constraints

Besides using occurrence statements referring to specific situations and histories, Pinto also introduces a class of action occurrences that apply to all paths in the tree of situations. To achieve this, Pinto introduces three predicates on situations. The predicate $legal_{poss} : \mathcal{S}$ represents the fact that all actions on the path leading to a situation s are executable. This predicate is the same as *exec* in Reiter's work on successor state axioms [118] (see chapter 3, section 3.1). The predicate $proper : \mathcal{S}$ is used to channel constraints on legal situations. One such constraint is that the situation should be reachable via an executable action sequence:

$$proper(s) \Rightarrow legal_{poss}(s) \tag{5.23}$$

Then different types of occurrences can be implemented in terms of additional constraints on *proper*. One example is non-preventable occurrences in time such as "the sun will rise tomorrow at 6:03." This type of occurrence is represented using a predicate $occurs_{np} : \mathcal{T} \times \mathcal{A}$, which constrains *proper* as follows.

$$\begin{array}{l} proper(s) \Rightarrow \\ \quad [occurs_{np}(a,t) \wedge t < start(s) \Rightarrow hasOccurred(a,t,s)] \end{array} \tag{5.24}$$

That is to say, for any proper situation s, if the non-preventable event a is to occur at t and s starts later than t, then a must have occurred at t in the history of s. The statement about the sunrise can now be encoded as follows.

$$occurs_{np}(Sunrise, 6:03) \tag{5.25}$$

Other kinds of occurrences can be implemented by introducing new occurrence predicates and using them to constrain *proper*. Using predicate completion, the definition of *proper* can then be made biconditional.

$$\begin{array}{l} proper(s) \equiv \\ \quad [legal_{poss}(s) \wedge \\ \quad \forall a,t[occurs_{np}(a,t) \wedge t < start(s) \Rightarrow hasOccurred(a,t,s))] \wedge \\ \quad \text{Constraints for other occurrence predicates...} \quad] \end{array} \tag{5.26}$$

Finally, a situation s is considered to be legal with respect to the action preconditions and occurrence statements if at every time after its start there is a proper future history that contains s.

$$\begin{array}{l} legal(s) \Rightarrow \\ \quad \forall t[start(s) < t \Rightarrow \exists s'[s \prec s' \wedge t < start(s') \wedge proper(s')]\,] \end{array} \tag{5.27}$$

Yet, it is not really clear how one can actually prove that a situation is legal. We can see two potential problems. First, the timing of situations and actions might vary between different models. For instance, given the sunrise statement (5.25), is $result(\{DrawCard\}, S0)$ a legal situation or not? This depends on whether

$$start(result(\{DrawCard\}, S0)) < 6{:}03$$

or

$$6{:}03 \leq start(result(\{DrawCard\}, S0)).$$

In the first case, the situation is legal, and in the second case, it is not; instead,

$$result(\{DrawCard\}, result(\{Sunrise\}, S0)$$

would be legal. This problem can probably be avoided by using timed action terms, as in $result(\{DrawCard(5{:}00)\}, S0)$. The second problem is that the occurrence predicates (e.g. $occurs_{np}$) are under-specified. For instance, given the axioms above, one cannot prove that specific action occurrence statements, such as

$$occurs_{np}(Sunrise, 4{:}00),$$

do not hold. Consequently, even if we time *DrawCard* to 5:00, we cannot prove that the situation

$$result(\{DrawCard(5{:}00)\}, S0)$$

is legal, as this would require proving

$$\neg occurs_{np}(Sunrise, t)$$

for all $t \leq 5{:}00$. This problem might be avoided by minimizing the different kinds of occurrence predicates, but no such minimization is suggested by Pinto.

5.3.3 Triggered action occurrences

As already mentioned, Pinto addresses several types of occurrences. One type is non-preventable occurrences, like in the sunrise example (5.25). We will also discuss another type: triggered occurrences. The example provided

by Pinto is "if you eat the forbidden fruit, you will be expelled." Triggered occurrences are modeled with a predicate $occurs_{to} : \mathcal{A} \times \mathcal{T} \times \mathcal{S}$:

$$\begin{array}{l} EatFruit \in c \Rightarrow \\ \quad occurs_{to}(Expel, start(do(c,s)) + \Delta, do(c,s)) \end{array} \tag{5.28}$$

thus, if $EatFruit$ is part of a concurrent action, then $Expel$ always occur Δ time-units later.

Like $occurs_{np}$, the new predicate $occurs_{to}$ constrains the set of proper situations. The following necessary condition on *proper* defines how this is done.

$$\begin{array}{l} proper(s) \Rightarrow \\ \quad [occurs_{to}(a,t,s') \wedge t < start(s) \wedge s' \prec s \Rightarrow \\ \quad\quad hasOcurred(a,t,s)] \end{array} \tag{5.29}$$

We shall return to this example in paper III, in the context of TAL.

5.3.4 Conclusions

In his more recent work, Pinto still makes a distinction between hypothetical, branching-style information and non-hypothetical ("actual") narrative-style information. Narrative information is represented using occurrence statements, and applies to all branches. Thus, we have gone from reasoning about one possible development (i.e. the actual path) of a narrative to multiple alternative developments of a narrative. The alternatives are in terms of other actions that might occur besides those in the narrative, but does not relate to e.g. different initial states. For instance, the following are the action occurrences in the coffee machine scenario.

$$Occurs_{np}(DrawCard, 2.4) \wedge Occurs_{np}(PressButton, 5.0). \tag{5.30}$$

Given this narrative, it is possible to reason hypothetically about how this narrative would develop depending on what other actions occur. One can for instance hypothesize that somebody resets the machine and thereby removes the credit between 2.5 and 5.0, and conclude that then there would be no coffee.

$$\begin{array}{l} Holds(alive, result(\{PressButton\}, result(\{Reset\}, \\ \quad result(\{DrawCard\}, S0)))) \wedge \\ legal(result(\{PressButton\}, result(\{Reset\}, \\ \quad result(\{DrawCard\}, S0)))) \end{array} \tag{5.31}$$

Note that although it is possible to reason hypothetically about what is not specified narrative-wise, the narrative part is not accessible to hypothetical (that is counter-factual) reasoning.[4]

The representation of occurrences has clearly been enriched: the different occurrence types enumerated in the beginning of this section constitute a significant contribution to extending the ontology of SC (and indeed of any logic of action and change). What are missing are well-defined non-monotonic policies for establishing when the different types of occurrences do not occur.

5.4 Narrative-based approaches

There have also been attempts to start from a narrative-based approach and add branching-time features. One example is Pelavin's logic [109] presented in chapter 4. Another example is Sandewall's branching time, presented in his book *Features and Fluents* [124] as an alternative to linear time. As a matter of fact, the logics in the book (including the PMON logic from which TAL is derived) are defined both for the linear and branching cases. The idea behind the branching temporal structure is that time-points are complex objects that contain not only metric information, but also encode the history of events that have occurred up to the current time point. A time-point is a tuple $\langle t, E\rangle$ where t is a natural number and E is the set of occurrences that have been initiated so far. For instance, $\langle 0, \emptyset\rangle$ is the initial time-point; $\langle 12, \emptyset\rangle$ is the 12th time-point in a branch where no actions have yet occurred; and $\langle 12, \{\langle 4, \mathsf{DrawCard}\rangle, \langle 7, \mathsf{PressButton}\rangle\}\rangle$ is the 12th time-point in a branch where the action DrawCard was started at 4 and PressButton was started at 7.

As mentioned at the beginning of this chapter, there have been many proposals on how to encode narratives in situation calculus. Provetti [117] presents an extension of the event calculus which mixes features of linear metric and situation calculus-style branching time. There is a linear time line on which the actual events occur (e.g. $Happens(DrawCard, 5)$) and where actual facts hold (e.g. $HoldsAt(credit, 6)$). In addition, each time point t is associated with a situation $Sit(t)$. From each situation, one can reason hypothetically situation calculus-wise using a function $Res(a, s)$ and

[4] Compare this to Reiter's concurrent temporal SC [120] (see section 5.2). If we consider Reiter's action sequences to be a kind of narratives, then Reiter's SC permits reasoning about alternative narratives. In paper V, we present a logic based on TAL that permits reasoning about alternative narratives and alternative developments relative to a specific narrative.

a predicate $HypHolds(f, s)$. For instance, assume that the agent actually draws the card: $Happens(DrawCard, 5)$. From this, one can reason what the result would be if the agent presses the button:

$$HypHolds(\textit{has-coffee}, Res(PressButton, Sit(5))).$$

The resulting structure is a linear metric time-line, where each time-point is the root of a branching event-driven structure. Apparently, a weakness of this approach is that one cannot do hypothetical reasoning involving metric time; that is restricted to the actual time line.

5.5 Summary

This chapter presents three proposals — two by Pinto and one by Reiter — for extending SC with narrative-style features, in order to support both hypothetical reasoning and reasoning about explicit time. One observes that although these proposals represent significant advancements for the temporal expressivity of situation calculus, they do not reach the temporal expressiveness one associates with narrative-based languages such as TAL. In particular, they assume that the timing of and order between actions is completely specified. The reason is that in all these proposals, the linear time-line is secondary to the branching event-driven temporal structure. Consequently, time is still not independent from actions, and actions can be temporally related only in very limited ways. In paper V, we present an alternative approach that has separated the linear time-line from actions. This approach permits reasoning hypothetically about narratives with actions with only partially specified timing, as well as about alternative consequences of a single narrative. In its separation of action occurrences from the time line, this approach lies closer to Pelavin's logic (see section 4.4 in chapter 4) than to the extensions of SC presented in this chapter.

Both Reiter and Pinto address the phenomena of triggered actions and events. In particular, Pinto addressed preventable, triggered and partially specified occurrences. In paper III, we study these topics in the context of TAL.

Part II
Papers

Comments on the papers

The following comprises comments on the papers presented in part II, and how they relate to each other and to the presentation in part I.

Paper I

The first paper, *Reasoning about concurrent interaction* [65], was published in the *Journal of Logic and Computation* in 1999 and presents a logic, TAL-C, for the representation of concurrent actions and their effects. It is a joint work with Joakim Gustafsson. An earlier version [64] was published in the *Linköping University Electronic Press*, in November 1997 under the title *Reasoning about actions in a multi-agent environment.* The two versions differ in notation (in particular in the names of influences) and the discussion about modularity did not appear in the 1997 version.

As the subject of the first paper is concurrent interaction, *Chapter 4: Concurrent interaction*, is of obvious relevance. The purpose of chapter 4 is to provide a more in-depth presentation and analysis of some particularly interesting approaches to concurrency; this was absent in the journal article due to page restrictions. The contribution of this article relative to the approaches presented in chapter 4 can be summarized as follows:

- TAL-C can represent a wide range of interactions, including the effects of one action interfering with or providing the conditions of another action; synergistic effects; conflicting and cumulative effect interactions; and interacting conditions in terms of resource conflicts. The most advanced of the other approaches in this respect are Pinto [112] (resources, effect cancellation and effect synergy), and Pelavin [109] (cumulative effects, resource constraints, conflicting effects[5]).

[5] In Pelavin's terminal example. See section 4.4.5.

- Interactions are not restricted to actions and their direct effects, but can also involve causal dependencies. Only Georgeff [45] and Pinto in his later paper [113] offer this possibility, by encoding causal dependencies as triggered events.

- Actions have extension, and what happens during an action is as important as what holds before the action is executed. Of the others, only Pelavin addresses actions with extension.

- There is a notion of explicit time; of the others, only Pelavin's approach uses explicit time.

- Additions and modifications in a TAL-C narrative are local operations that preserve modularity. Modularity issues have also been addressed by Pelavin [109], and Pinto in his later paper [113], but not in the same systematic fashion as in TAL-C.

- TAL-C has a first-order semantics and proof theory (apart from the axioms for the temporal domain; see appendix A). In particular the approach of Pelavin [109] is different in this respect: Pelavin relies on a fairly complex non-standard semantics.

Paper II

The second paper, *Delayed effects of actions* [66], was presented at ECAI in Brighton, 1998, and is a joint work with Joakim Gustafsson and Patrick Doherty. It is a continuation of paper I, and some of the contents of paper I are repeated, although in a more compact form. The contribution of paper II is that it addresses the representation of effects that occur some time after the execution of an action (i.e. delayed effects), and processes that might be initiated by an action but then continue after the action has ended. That is a problem that has not received much attention in RAC; some exceptions are Sandewall [123], Shanahan [128] and Doherty and Gustafsson [28]. However, none of these earlier approaches had any means for dealing with concurrent interactions, which might occur between an action and a delayed effect or process, or between two processes. These complications are central to paper II.

Paper III

The third paper, *Causal relations between actions in TAL*, is an unpublished manuscript. It concerns the representation of causal relations between actions, and covers actions triggered by certain conditions in the world, actions triggered by other actions, and actions prevented by other actions. In part I of the thesis, chapter 5, and in particular section 5.3, are relevant. The main contributions of this paper are (1) the fact that actions have extension[6] (2) the treatment of indeterminate time, and (3) the simple circumscription policy.

Paper IV

The fourth paper, *Reasoning about incomplete initial information and nondeterminism in situation calculus* [62], was presented at IJCAI in Nagoya in 1997. It is the only paper that is not about extending TAL. Instead, it is about extending the situation calculus. The paper was motivated by the observation made in chapter 3, section 3.1.7, that it is not possible to make statements about alternative consequences of an action sequence in SC(R). It is shown how this affects the possibility of doing deductive planning. The solution proposed in paper IV is to adopt the PMON minimization policy [124] for representing change; and to view situations as states, as done by Baker (see section 3.1.2 in chapter 3). In general, section 3.1 in chapter 3 is relevant, and also section 3.7.1, about dynamic logic, can be of interest for comparisons.

Paper V

The fifth paper, *Anything Can Happen: on Narratives and Hypothetical Reasoning* [63], was presented at KR in Trento, 1998. It has a theme similar to that of paper IV: reasoning about alternative courses of action and about alternative consequences of a single course of action. However, the notion of a course of action is much richer here; instead of a sequence of actions, it is a narrative with a metric time-line, and the timing of actions in the narrative can be indeterminate. Consequently, it can be viewed as an attempt to combine narrative-based and branching-time features in the sense

[6] This is not new to TAL, but has not previously been treated in the context of triggered actions.

discussed in chapter 5, *Enriching the representation of narratives.* The approaches (Pinto, Reiter) in that chapter do not support reasoning about alternative consequences of a single narrative, though, and they further assume that the timing of actions is completely specified in the narrative. The logic presented in this paper, the narrative logic (NL), does not have these limitations. In addition, NL supports gradual extension of narratives, and narratives can be combined together in different manners. This makes NL a suitable formalism for representing plan synthesis and other reasoning tasks. There are connections to section 4.4 in chapter 4, which describes Pelavin's logic. Also section 3.7.2, concerning temporal logics, can be of interest for comparisons.

Paper I

Reasoning about Concurrent Interaction

Lars Karlsson and Joakim Gustafsson
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
Tel: +46-13-28 24 28, Fax: +46-13-28 26 66
Email: {larka,joagu}@ida.liu.se

Abstract

In this paper we present TAL-C, a logic of action and change for worlds with action concurrency. TAL-C has a first-order semantics and proof theory. It builds on an existing logic TAL, which includes the use of dependency laws for dealing with ramification. It is demonstrated how TAL-C can represent a number of phenomena related to action concurrency: action duration, how the effects of one action interfere with or enable another action, synergistic effects of concurrent actions, conflicting and cumulative effect interactions, and resource conflicts. A central idea is that actions are not described as having effects that directly alter the world state. Instead, actions produce influences, and the way these influences alter the world state are described in specialized influence laws. Finally, we address how TAL-C narratives can be written to support modularity.

1 Introduction

To an agent operating in a complex multi-agent environment, it is important to be able to reason about how the environment changes due to the actions that the agents perform. Most of the work in reasoning about action and change has been done under the (sometimes implicit) assumption that there is a single agent that performs sequences of actions. This is a comparatively easier problem than reasoning about action and change under a multi-agent assumption, or under a non-sequentiality assumption for a single agent, which by necessity introduces the complication that one or several agents can perform actions concurrently. Concurrency, in turn, can involve a wide range of interactions between actions, which makes it unlikely that there is one single, uniform technique of general applicability.

1.1 TAL-C

This paper presents an approach to reasoning about action and change which supports the description of concurrent actions with nontrivial interactions. The approach is based on TAL (*Temporal Action Logic*, formerly called PMON) which in its original form [124, 26] covers worlds with a natural numbers time domain and sequentially executed actions whose effects can be context-dependent (different initial states can lead to different effects) and nondeterministic (the same initial state can lead to several different effects). From the perspective of concurrency, an important property of TAL is that actions have durations (that is occur over an interval of time). TAL has recently been extended to support the description of ramifications, or indirect effects [48], and to deal with qualified action descriptions [30]. In this paper we present TAL-C, a development of the TAL/PMON formalism, which combines ramification and concurrency. Parts of the results in this paper have also been exploited for dealing with delayed effects [66]. In TAL-C, the description of concurrent interactions are done on the level of features (state variables). The central idea is that actions are not modeled to directly change the world state, but to produce influences on features. For each feature one can then use influence laws to specify how it is affected by its various associated influences. Besides providing a flexible and versatile means for describing concurrent interactions, the use of influences and influence laws permits describing the properties of actions, dependencies and features in isolation, thereby supporting modularity. The major merit of TAL-C is that it combines (a) a standard first-order semantics and proof theory; (b) a notion of explicit time, which makes it possible to reason

about the durations and timing of actions and effects; and (c) modeling of a number of important phenomena related to concurrency, in addition to ramification and nondeterminism.

1.2 The two language levels of TAL-C

TAL-C, like its predecessors, is a formalism that consists of two languages. First, there is the surface language $\mathcal{L}(ND)$ which provides a number of macros that make it possible to describe TAL-C narratives in a concise way. Scenarios, or narratives, in $\mathcal{L}(ND)$ are translated to the standard first-order language $\mathcal{L}(FL)$ by expanding these macros, and standard first-order deduction can then be used for reasoning about the narratives. $\mathcal{L}(ND)$ is used throughout most of this paper; the translation to $\mathcal{L}(FL)$ is presented in appendix A. Therefore, some definitions are initially presented in $\mathcal{L}(ND)$ but are later encoded as macro expansions in the translation to $\mathcal{L}(FL)$.

1.3 Organization of the paper

The rest of the paper is organized as follows. In section 2, TAL is introduced with an example and a definition of the syntax of the surface language $\mathcal{L}(ND)$. In section 3, a number of interesting cases of concurrency are identified and discussed. Section 4 outlines an approach to concurrency and introduces the concept of influences, and section 5 introduces the extensions to TAL that make TAL-C. In section 6, the means for solving the problems identified in section 3 are developed. Section 7 addresses modularity in the context of TAL-C. Section 8 provides an overview of previous work on concurrency, and section 9 contains some conclusions. Finally, appendix A presents a translation from the surface language $\mathcal{L}(ND)$ to the first-order language $\mathcal{L}(FL)$.

2 Preliminaries

In TAL, the world consists of objects, features that have time-variant values, and actions that can be executed by agents and that affect the values of features over time. A TAL narrative is a structured collection of statements referring to the dynamics of the world in terms of action laws and dependency laws, action occurrences and their timing, and observations of feature values at different time-points. A narrative encodes a specific course of actions (other examples of narrative-based formalisms are Event Calculus

[69, 58] and Allen's logic [6]; see also Karlsson [63] on representing TAL narratives as first-order objects). The surface language $\mathcal{L}(ND)$ for sequential narratives is presented in this section, and extensions for concurrency are introduced in section 5.

2.1 Narratives in TAL

The following is a narrative in $\mathcal{L}(ND)$. It describes a world with two types of actions (LightFire and PourWater), and a number of agents (bill and bob) and other objects (wood1). For notational convenience, all variables appearing free are implicitly universally quantified.

$$
\begin{array}{ll}
\text{acs1} & [s,t]\mathsf{LightFire}(a,x) \Rightarrow \\
 & \quad ([s]\mathsf{dry}(x) \wedge \mathsf{wood}(x) \Rightarrow R((s,t]\mathsf{fire}(x))) \\
\text{acs2} & [s,t]\mathsf{PourWater}(a,x) \Rightarrow (R([s,t]\neg\mathsf{dry}(x)) \wedge \\
 & \quad ([s]\mathsf{fire}(x) \Rightarrow R((s,t]\neg\mathsf{fire}(x)))) \\
\text{obs1} & [0]\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}) \\
\text{occ1} & [2,5]\mathsf{LightFire}(\mathsf{bill},\mathsf{wood1}) \\
\text{occ2} & [6,7]\mathsf{PourWater}(\mathsf{bob},\mathsf{wood1})
\end{array}
\tag{1}
$$

Notice that all lines are labeled. The labeling reflects the structure of the narrative; different types of statements serve different purposes.

Acs1 and *acs2* are action laws, which describe the effects of specific action types under different conditions. The first action law states that if an agent a lights a fire using some wood x, and if the wood is dry, then the result will be that the wood is on fire. The expression $[s,t]\mathsf{LightFire}(a,x)$ denotes the action in question where $[s,t]$ is its time interval, and $[s]\mathsf{dry}(x) \wedge \mathsf{wood}(x)$ denotes that the features $\mathsf{dry}(x)$ and $\mathsf{wood}(x)$ hold at time-point s. The statement $R((s,t]\mathsf{fire}(x))$ denotes that the feature $\mathsf{fire}(x)$ is reassigned to become true somewhere in the interval $(s,t]$, and in particular that it is true at the last time-point t of the interval.[1] The terms s and t are time-point variables, and are assumed to be universally quantified (as are a and x). The second action law states that if somebody pours water on an object, then the object will no longer be dry, and will cease being on fire.

Obs1 is an observation statement. It states that the wood (denoted wood1) is dry and not burning at the initial time-point 0. Observations are assumed to be correct, and can refer to arbitrary time-points or intervals; in the latter case, the notation $[\tau,\tau']\phi$ is used (e.g. $[0,6]\mathsf{dry}(\mathsf{wood1})$). *Occ1*

[1] Previously, the notation $[s,t]\mathsf{fire}(x) := T$ has been used for reassignment [124]. However, in order to be coherent with the notation for the additional operations on features that are introduced in this paper, $R((s,t]\mathsf{fire}(x))$ has been preferred.

and *occ2* are occurrence statements. They describe what actions actually occur in a narrative. A fire is lit by the agent bill during the temporal interval $[2, 5]$, and then the agent bob pours water on the wood during the temporal interval $[6, 7]$. No actions besides those explicitly appearing in the occurrence statements are assumed to occur in the narrative. By using non-numerical temporal constants, such as $[\mathsf{s}_1, \mathsf{t}_1]\mathsf{LightFire}(\mathsf{bill}, \mathsf{wood1})$, the exact timing of an action can be left unspecified.

It is possible for features to have domains other than boolean truth values. In this case, the notation $\phi \hat{=} \omega$ is used to state that the feature ϕ has the value ω, like in the statement $[5]\mathsf{traffic\text{-}light} \hat{=} \mathsf{green}$. The same notation is applicable (but optional) to booleans, e.g. $[s]\mathsf{dry}(x) \hat{=} \mathsf{T}$.

It is commonly recognized that in domains where there are more complex dependencies between features, specifying all possible direct and indirect effects of an action is not feasible. The problem of specifying all effects of actions in a compact manner is called the ramification problem [47]. In TAL, dependency laws are used to deal with this problem (for other approaches to ramification, see e.g. [87, 136, 93]). With dependency laws, one can specify general relations between features once, rather than having to repeat them in each relevant action law.[2] The following are two examples that could complement the description of the fire lighting action in (1). *Dep1* states that if an object starts burning, then it also starts smoking, and *dep2* states that if an object starts smoking and the damper is closed (or the object is smoking and the damper becomes closed) then the agent's eyes become sore. The C_T operator denotes that the expression inside was false at the previous time point and has just become true: $C_T([t]\alpha) =_{def} (\forall t'[t = t' + 1 \Rightarrow [t']\neg\alpha] \wedge [t]\alpha)$. Again, free variables are implicitly universally quantified.

$$\begin{array}{ll} \text{dep1} & C_T([t]\mathsf{fire}(x)) \Rightarrow R([t]\mathsf{smoking}(x)) \\ \text{dep2} & C_T([t]\mathsf{smoking}(x) \wedge \neg\mathsf{damper\text{-}open}) \Rightarrow R([t]\mathsf{eyes\text{-}sore}(a)) \end{array} \tag{2}$$

Reassignment (R) plays an important role in the solution to the frame problem [97] in TAL. Reassignment expresses change, and unless a feature is involved in reassignment, it is assumed not to change. The reassignment

[2] Note that there are restrictions as to how dependency laws can be combined in TAL, in order to avoid cycles of instantaneous dependencies that can result in "spontaneous triggering" of dependency laws. Theories that do not contain any dependency cycles are called stratified (see [48]).

operator is defined as follows,[3]

$$R((\tau,\tau']\alpha) =_{def} (X((\tau,\tau']\alpha) \wedge [\tau']\alpha) \quad (3)$$
$$R([\tau]\alpha) =_{def} (X([\tau]\alpha) \wedge [\tau]\alpha).$$

X is an operator that represents "occlusion" of the features in α, whereas the right-most parts of the definitions denote that α holds at the end of the interval. Occlusion represents an exception from the general principle of persistence, so features that are occluded are allowed to change from one time-point to the next. By minimizing occlusion, the time-points where a specific feature can change value are restricted to those time-points where the feature in question explicitly appears in a reassignment. The fact that features normally do not change is encoded in the no-change axiom below. It states that unless the feature f is occluded at time-point $t+1$, it must have the same value at $t+1$ as at time-point t.[4]

$$\forall t, f, v[\neg X([t+1]f) \Rightarrow ([t]f \hat{=} v \equiv [t+1]f \hat{=} v)] \quad (4)$$

To illustrate how one can reason in TAL[5], consider the narrative in (1). From timepoint 0 to 2, there is no reassignment and therefore no occlusion, so one can with the aid of (4) infer

$$[0,2]\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}).$$

From lines *acs1* and *occ1*, it follows that

$$[2]\mathsf{dry}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}) \Rightarrow R((2,5]\mathsf{fire}(\mathsf{wood1}))$$

holds, which yields $[5]\mathsf{fire}(\mathsf{wood1})$. As the two other features are not occluded, due to (4) one has that

$$[3,5]\mathsf{dry}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1})$$

holds, and then that

$$[6]\mathsf{dry}(\mathsf{wood1}) \wedge \mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1})$$

holds. From *acs2* and *occ2*, it follows that

$$R((6,7]\neg\mathsf{dry}(\mathsf{wood1})) \wedge R((6,7]\neg\mathsf{fire}(\mathsf{wood1}))$$

[3] The following definitions are actually encoded in the translation process from $\mathcal{L}(ND)$ to $\mathcal{L}(FL)$ in appendix A. The $\mathcal{L}(ND)$ versions in this section are preliminary.

[4] This version of the axiom is preliminary. See appendix A.

[5] The following is not a proof, it is an informal account based on the intuitions behind the surface language. Formal proofs are done using natural deduction in the base language (for an example of this, see the translation of narrative (15) in appendix A.3).

holds, which yields

$$[7]\neg\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}).$$

Finally, as nothing happens after 7,

$$[t]\neg\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1})$$

holds for all $t \geq 7$.

2.2 The language $\mathcal{L}(ND)$

This section defines the surface language $\mathcal{L}(ND)$ for sequential narratives. Extensions for concurrency are introduced in section 5, and the translation to the first-order language $\mathcal{L}(FL)$ is presented in appendix A. We use the overline as abbreviation of a sequence, when the contents of the sequence is obvious. For example, $f(\overline{x}, \overline{y})$ means $f(x_1, ..., x_n, y_1, ..., y_m)$.

Definition 1 (vocabulary) A TAL vocabulary $\nu = \langle C, F, A, T, V, R, S, \sigma \rangle$ is a tuple where C is a set of constant symbols, F is a set of feature symbols, A is a set of action symbols, T is a set of temporal function symbols, V is a set of value function symbols, R is a set of relation symbols, S is a set of basic sorts and σ is a function that maps each member of these symbol sets to a sort declaration of the form $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ or $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n \rightarrow \mathcal{S}_{n+1}$ where $\mathcal{S}_i \in S$.

Definition 2 (basic sorts) There are a number of sorts for values $\mathcal{V}_i$, including the boolean sort $\mathcal{B}$ with the constants $\{\mathsf{T}, \mathsf{F}\}$. There are a number of sorts for features $\mathcal{F}_i$, each one associated with a value domain $dom(\mathcal{F}_i) = \mathcal{V}_j$ for some j, a sort for actions $\mathcal{A}$, and a temporal sort $\mathcal{T}$.

$\mathcal{T}$ is assumed to be an interpreted sort, but can be axiomatized in first-order logic as a subset of Presburger arithmetic [68] (natural numbers with addition).

Definition 3 (terms) A *value term* ω is a variable v or a constant v of sort $\mathcal{V}_i$ for some i, or an expression $\mathsf{g}(\omega_1, \ldots, \omega_n)$ where $\mathsf{g} : \mathcal{V}_{k_1} \times \ldots \times \mathcal{V}_{k_n} \rightarrow \mathcal{V}_i$ is a value function symbol and each ω_j is of sort $\mathcal{V}_{k_j}$. A *temporal term* τ is a variable t or a constant $0, 1, 2, 3, \ldots$ or $\mathsf{s}_1, \mathsf{t}_1, \ldots$, or $\tau_1 + \tau_2$, all of sort $\mathcal{T}$. A fluent term ϕ is a feature variable f or an expression $\mathsf{f}(\omega_1, \ldots, \omega_n)$ where $\mathsf{f} : \mathcal{V}_{k_1} \times \ldots \times \mathcal{V}_{k_n} \rightarrow \mathcal{F}_i$ is a feature symbol and each ω_j is of sort $\mathcal{V}_{k_j}$.

Definition 4 (temporal and value formulae) If τ, τ' are temporal terms, then $\tau = \tau'$, $\tau < \tau'$ and $\tau \leq \tau'$ are *temporal formulae.* A *value formula* is of the form $\omega = \omega'$ where ω and ω' are value terms, or $\mathsf{r}(\omega_1, \ldots, \omega_n)$ where $\mathsf{r} : \mathcal{V}_{k_1} \times \ldots \times \mathcal{V}_{k_n}$ is a relation symbol and each ω_j is of sort $\mathcal{V}_{k_j}$.

Definition 5 (fluent formula) An *elementary fluent formula* has the form $\phi \hat{=} \omega$ where ϕ is a fluent term of sort $\mathcal{F}_i$ and ω is a value term of sort $dom(\mathcal{F}_i)$. A *fluent formula* is an elementary fluent formula or a combination of fluent formulae formed with the standard logical connectives and quantifiers.

The elementary fluent formula $\phi \hat{=} \mathsf{T}$ can be abbreviated ϕ.

Definition 6 (timed formulae) Let τ, τ' be temporal terms and α a fluent formula. Then $[\tau, \tau']\alpha$, $(\tau, \tau']\alpha$ and $[\tau]\alpha$ are *fixed fluent formulae*, $C_T([\tau]\alpha)$ is a *becomes formula*, $R((\tau, \tau']\alpha)$, $R([\tau, \tau']\alpha)$and $R([\tau]\alpha)$ are *reassignment formulae*, and $X((\tau, \tau']\alpha)$, $X([\tau, \tau']\alpha)$ and $X([\tau]\alpha)$ are *occluded formulae.*

Definition 7 (static formula) A logical combination (including quantifiers) of temporal and value formulae, fixed fluent formulae and/or becomes formulae is called a *static formula.*

Definition 8 (change formula) A *change formula* is a formula that has (or is rewritable to) the form $\mathcal{Q}\overline{v}(\alpha_1 \vee ... \vee \alpha_n)$ where $\mathcal{Q}\overline{v}$ is a sequence of quantifiers with variables, and each α_i is a conjunction of static, occlusion and reassignment formulae. The change formula is called *balanced* iff the following two conditions hold. (a) Whenever a feature $f(\overline{\omega})$ appears inside a reassignment or occlusion formula in one of the α_i disjuncts, then it must also appear in all other α_i's inside a reassignment or occlusion formula with exactly the same temporal argument. (b) Any existentially quantified variable v in the formula, whenever appearing inside a reassignment or occlusion formula, only does so in the position $\phi \hat{=} v$.

Definition 9 (application formula) An *application formula* is any of the following: (a) a balanced change formula; (b) $\Lambda \Rightarrow \Delta$, where Λ is a static formula and Δ is a balanced change formula; or (c) a combination of elements of types (a) and (b) formed with $\wedge$ and $\forall$.

Definition 10 (occurrence formula) An *occurrence formula* has the form $[\tau, \tau']\Phi(\overline{\omega})$, where τ and τ' are elementary time-point expressions, Φ is an action name of sort $\mathcal{V}_1 \times \ldots \times \mathcal{V}_n \rightarrow \mathcal{A}$ and the value terms in $\overline{\omega}$ are of matching sorts.

Definition 11 (narrative components) An action law (labeled *acs*) has the form $\forall t, t', \overline{x}, \overline{y}[[t, t']\Phi(\overline{x}) \Rightarrow \Psi(\overline{x}, \overline{y})]$ where $[t, t']\Phi(\overline{x})$ is an occurrence formula and $\Psi(\overline{x}, \overline{y})$ is an application formula. A dependency law (labeled *dep*) has the form $\forall t, \overline{x}[\Psi(\overline{x})]$ where $\Psi(\overline{x})$ is an application formula. An observation (labeled *obs*) is a static formula. An occurrence (labeled *occ*) is an occurrence formula $[\tau, \tau']\Phi(\overline{\omega})$ where τ, τ', $\overline{\omega}$ all are variable-free terms.

3 Variations on the concurrency theme

In this section, we release the sequentiality assumption from the previous section and identify a number of issues that a language for narratives with concurrency should be able to handle. We observe that TAL is sufficient for handling some of these issues, namely action duration and concurrent execution of independent actions. When it comes to interacting actions, we observe that TAL is not sufficient in a number of cases. This observation forms the basis for the discussion on how to extend TAL to handle concurrency in the subsequent sections. Note that although we address mainly actions, the discussion applies also to dependencies.

In many domains, it is a fact that actions take time. As long as actions occur sequentially, the only way actions can interact is when the effects of one action affect the context in which a later action is executed. Therefore, it can make sense to abstract away action durations in the case of sequentiality, as is done in for instance basic situation calculus [118]. However, in the case of concurrency, the durations of actions are important for a number of reasons. First, the way the durations of two or more actions overlap can determine how they interact. Second, an action can overlap with two or more other actions without these latter actions overlapping. Third, what happens in the duration of an action can be important for how it interacts with other concurrent actions. TAL has explicit time and actions in TAL have duration.

Concurrent actions can be independent, and involve disjoint sets of features. In this case, the combined effect of the concurrent actions is simply the union of the individual effects, as in the example below.

$$
\begin{array}{ll}
\text{acs1} & [s,t]\mathsf{LightFire}(a,x) \Rightarrow \\
 & \quad ([s]\mathsf{dry}(x) \wedge \mathsf{wood}(x) \Rightarrow R((s,t]\mathsf{fire}(x))) \\
\text{acs2} & [s,t]\mathsf{PourWater}(a,x) \Rightarrow R((s,t]\neg\mathsf{dry}(x)) \wedge R((s,t]\neg\mathsf{fire}(x)) \\
\text{obs1} & [0]\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}) \\
\text{obs2} & [0]\mathsf{dry}(\mathsf{wood2}) \wedge \neg\mathsf{fire}(\mathsf{wood2}) \wedge \mathsf{wood}(\mathsf{wood2}) \\
\text{occ1} & [2,7]\mathsf{LightFire}(\mathsf{bill},\mathsf{wood1}) \\
\text{occ2} & [2,7]\mathsf{LightFire}(\mathsf{bob},\mathsf{wood2})
\end{array}
\tag{5}
$$

Here TAL yields the conclusion $[7](\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{fire}(\mathsf{wood2}))$ as intended. Concurrency of independent actions does not pose a problem for TAL [139], nor should it for most other formalisms that do not rely on some kind of explicit frame axioms. The difficult problems arise when concurrent actions are not independent.

We address three different problems related to concurrent execution of interdependent actions. The first problem is due to the fact that the conditions under which an action is executed are not always stable, but may be altered by the effects of other concurrent actions. Consider a slight modification of (5), where bob pours water on the fire wood while bill is lighting the fire. The intuitive conclusion is that the wood should not be on fire at 7. We formalize this scenario in TAL by modifying some lines in the narrative (5) above.[6]

$$\begin{array}{ll} \text{occ1} & [2,6]\mathsf{LightFire}(\mathsf{bill},\mathsf{wood1}) \\ \text{occ2} & [3,5]\mathsf{PourWater}(\mathsf{bob},\mathsf{wood1}) \end{array} \tag{6}$$

The modified narrative allows us to infer that the wood is actually on fire: $[7]\mathsf{fire}(\mathsf{wood1})$. The reason is that the effect of the $\mathsf{LightFire}(\mathsf{bill},\mathsf{wood1})$ action is determined only by the state at time-point 2 whereas the wood does not become wet until time-point 5. Thus, just referring to the starting state in preconditions in action laws is apparently not sufficient. One needs to take into account that the conditions under which the action is executed may be altered by the direct and indirect effects of other actions while the action is going on.

The effects of one action on the conditions of another action need not always be harmful. Often, the execution of one action can enable the successful execution of another simultaneous action. For instance, turning the latch of a door might enable opening the door. Sometimes, the enabling is mutual, and two (or more) actions have synergistic effects.

A slight modification of the narrative above illustrates the second problem, which is due to the way effects of actions are represented with reassignment. Assume that the two last lines in (5) are replaced with the following.

$$\begin{array}{ll} \text{occ1} & [3,7]\mathsf{LightFire}(\mathsf{bill},\mathsf{wood1}) \\ \text{occ2} & [3,7]\mathsf{PourWater}(\mathsf{bob},\mathsf{wood1}) \end{array} \tag{7}$$

That is to say, the lighting and the pouring actions have the same duration. Now, from *acs1* and *occ1* one can infer the effect $[7]\mathsf{fire}(\mathsf{wood1})$ and from *acs2* and *occ2* one can infer the effect $[7]\neg\mathsf{fire}(\mathsf{wood1})$. Notice that these two

[6] We only present the modified or added lines in the subsequent narratives.

effects are both asserted to be direct and indefeasible. Thus, the narrative becomes inconsistent. The conclusion one would like to obtain is again that the wood is not on fire.

A variation of effect interaction is when several actions affect the same feature in a cumulative way, that is when the total effect of the actions is an aggregate of the individual effects. Reassignment represents changes to absolute values (although these values can be calculated relative to other values), and obviously, aggregation of absolute values is not very meaningful. For instance, consider the following narrative with a box of coins.

$$\begin{array}{ll} \text{acs1} & [s,t]\mathsf{TakeCoin}(a,b) \Rightarrow \\ & \quad ([s]\mathsf{coins}(b)\hat{=}(n+1) \Rightarrow R((s,t]\mathsf{coins}(b)\hat{=}n)) \\ \text{obs1} & [0]\mathsf{coins}(\mathsf{box1})\hat{=}2 \\ \text{occ1} & [2,3]\mathsf{TakeCoin}(\mathsf{bill},\mathsf{box1}) \\ \text{occ2} & [2,3]\mathsf{TakeCoin}(\mathsf{bob},\mathsf{box1}) \end{array} \tag{8}$$

Both *occ1* and *occ2* produce the effect $R((2,3]\mathsf{coins}(\mathsf{box1})\hat{=}1)$. Notice that this effect states that $\mathsf{coins}(\mathsf{box1})$ has the absolute value 1, and not that $\mathsf{coins}(\mathsf{box1})$ changes by 1. Therefore, TAL yields $[3]\mathsf{coins}(\mathsf{box1})\hat{=}1$, while the intuitive conclusion is that there are 0 coins in the box at time-point 3.

The third problem is that the conditions for two concurrent actions might interfere. In particular, actions might compete for such things as space, objects and energy. For instance, if lighting a fire requires the use of two hands, then a two-handed agent is not able to light two fires concurrently. One way of addressing this type of conflicts is in terms of limited resources.

4 From action laws to laws of interaction

Based on the observations in the previous section, we argue that the way action laws are formulated in TAL (and many other formalisms) is not appropriate in the case of concurrency. In this section, two potential solutions are discussed. The first solution is to deal with interactions on the level of actions by allowing more expressive action laws. The second solution is to deal with interactions on the level of effects and features, by expressing effects in a less direct and absolute manner than direct reassignment of a feature. One can also imagine numerous combined approaches, for instance where conflicts can be detected on the level of features and then resolved on the level of actions, but that will not be discussed here.

4.1 Interactions on the level of actions

As the description of the effects of actions is usually centered around actions in most formalisms, it might seem like an obvious approach to also deal with concurrent interactions on the level of actions. This approach can be realized with action laws that refer to combinations of action occurrences, as in the work of Baral and Gelfond [11], Li and Pereira [81], Bornscheuer and Thielscher [18], and Reiter [120]. The following action laws, encoded in a hypothetical extended version of TAL, illustrate how one can overcome the problems in (6) and (7). Observe how *acs1* cancels the effect of LightFire in the presence of PourWater.

$$\begin{array}{ll} \text{acs1} & [s,t]\mathsf{LightFire}(a,x)\wedge \\ & \neg\exists a',s',t'[(s\leq s'<t\vee s<t'\leq t)\wedge \\ & \quad [s',t']\mathsf{PourWater}(a',x)]\Rightarrow \\ & \quad ([s]\mathsf{dry}(x)\wedge\mathsf{wood}(x)\Rightarrow R((s,t]\neg\mathsf{fire}(x))) \\ \text{acs2} & [s,t]\mathsf{PourWater}(a,x)\Rightarrow R((s,t]\neg\mathsf{dry}(x))\wedge R((s,t]\neg\mathsf{fire}(x)) \end{array} \tag{9}$$

Unfortunately, this solution has a number of weaknesses. It is no longer possible to describe actions in isolation, which weakens the case for modularity in action descriptions. All potentially interacting action combinations have to be identified and explicitly put into action laws. If actions have duration, the number of such combinations is not just determined by the number of actions, but also by the number of ways two or more actions can overlap in time.

Other factors also contribute to additional complexity. If an action has several effects, then one might not want an interference regarding just one feature to neutralize all the effects of the action. Furthermore, if the action interfered with starts before the interfering action, then the parts of the effects of the former action that are defined to occur before the starting time of the latter action should not be prevented, as this would imply causality working backwards in time.

An additional problem is that interactions are not confined to occur exclusively between the direct effects of actions, but might also involve indirect effects. Taking into consideration all potential interactions between combinations of actions and dependency laws would simply not be feasible for most nontrivial domains. Further, this would multiply all the previously mentioned complications.

4.2 Interaction on the level of features

As an alternative to an action-centered approach, we propose an approach based on the assumption that interactions resulting from concurrency are best modeled on the level of features, and not on the level of actions. The central ideas are as follows.

1. Actions provide an interface between the agent and the environment.

2. An action law does not explicitly encode the immediate effects that the action has on the state of the world. Instead, action laws encode what *influences* the action brings upon the environment. For instance, instead of stating $R((s,t]\mathsf{fire}(x))$ as an effect of the action $[s,t]\mathsf{LightFire}(a,x)$, one states $I((s,t]\mathsf{fire}^*(x,\mathsf{T}))$ where $\mathsf{fire}^*(x,\mathsf{T})$ represents an influence to make the feature $\mathsf{fire}(x)$ true (the I operator is similar to R, but denotes that the expression inside is true throughout the interval). An influence represents an inclination for a feature to take on a certain value or to change in a certain direction. Thus, it is more correct to consider action occurrence statements as representing action attempts that might fail to have their expected effects due to external interference rather than representing successfully executed actions.

3. Similarly, dependencies are modified to result in influences rather than actual change.

4. The actual effects that these influences (and indirectly the actions that caused them) have on the environment are then specified in a special type of dependency laws called influence laws. For instance, $[t]\mathsf{fire}^*(x,\mathsf{T}) \Rightarrow R([t]\mathsf{fire}(x))$. Generally, each feature is associated with a number of different influences. The behavior of a feature can be specified in a number of influence laws that describe how this feature is affected by different individual influences and combinations of influences. Thus, descriptions of features and how they change due to influences play a central role in TAL-C.

This approach implies that the emphasis of the world description has shifted from actions to features, and from action laws to influence laws. Further, it can be realized with a minimum of modifications of the TAL language, and in particular the first-order nature of TAL can be retained (see appendix A). Influence laws have the same form as dependency laws, which are already an integral part of the language, and influences can actually be represented

as features. The difference between influences and other features is purely conceptual; no new syntactic or semantic constructions particular to influences are required (although some new constructs applicable to features in general will be introduced). The term "actual features" will be reserved for features that are not influences.

Note that the use of influences as intermediaries of change serves two purposes. First, it makes it possible to avoid logical contradiction when two or more actions and dependencies affect the same feature. For instance, the combination $[5]\mathsf{fire}^*(\mathsf{wood1},\mathsf{T})$ and $[5]\mathsf{fire}^*(\mathsf{wood1},\mathsf{F})$ is logically consistent, but the combination $[5]\mathsf{fire}(\mathsf{wood1})$ and $[5]\neg\mathsf{fire}(\mathsf{wood1})$ is not. But there is more to concurrent interactions than preserving consistency. The actual outcomes of interactions need to be represented somehow, and a rich phenomenon like concurrent interactions requires a flexible representation that goes beyond the most stereotypical cases. This is supported in TAL-C by the fact that influences are first-order objects, which can be referred to in influence laws.

The use of influences in TAL-C has some resemblance to the way physical systems are often modeled. For instance, in mechanics the position/speed of a physical body is influenced by forces, and in hydraulics the levels of/flows between tanks are influenced by pressures. Note that in physics, influences often behave cumulatively. For instance, several forces can influence an object in a mechanical system at the same time, and these forces can be aggregated using vector addition. The term "influences" is explicitly used in qualitative reasoning about physical systems by (among others) Forbus [42]. For instance, in Forbus's qualitative process theory, the expressions I+(amount-of(dest),A[flow-rate]) and I-(amount-of(source),A[flow-rate]) denote that the flow rate between two interconnected tanks influences the amount of liquid in the tanks to increase respectively decrease. If a tank t has several pipes connected, then amount-of(t) will be subject to several influences. The flow rate is in turn proportional to the difference of pressure in the two tanks (flow-rate $\propto_{Q+}$ A[pressure(src)]$-$A[pressure(dest)]). Yet, the use of influences in TAL-C is not identical to the use in physical modeling. Although influences surely can be used in TAL-C in ways compatible with quantitative and qualitative physical modeling, they need not always faithfully represent actual physical entities or behave cumulatively.

5 Extending TAL to TAL-C

This section presents the differences between TAL and TAL-C. These include a new class of features, a new effect operator I and two new classes of narrative statements.

5.1 Persistent and durational features

In reasoning about action and change, features are typically considered to be persistent. That means that they only change under special conditions, such as during the execution of an action. In order to facilitate the use of influences, TAL-C is in addition equipped with a second type of features called durational features. These normally have a default value, and they can only have a non-default value under certain circumstances, such as during the execution of an action.[7] The predicate $Per(f)$ represents the fact that feature f is persistent, and $Dur(f, v)$ represents that f is durational with default value v. These predicates can be augmented with a temporal argument to support features with variable behavior, such as variable default value. The default behavior of persistent features is defined as follows.[8]

$$\forall t, f, v[Per(f) \Rightarrow (\neg X([t+1]f) \Rightarrow ([t]f \hat{=} v \equiv [t+1]f \hat{=} v))] \tag{10}$$

The default behavior of durational features is defined as follows,

$$\forall t, f, v\,[Dur(f, v) \Rightarrow (\neg X([t]f) \Rightarrow [t]f \hat{=} v)]. \tag{11}$$

Note that the distinction between persistent and durational features is in principle orthogonal to the distinction between actual features and influences, although in practice actual features are mostly persistent and influences are mostly durational. Naturally, one need not be confined to the two types of features presented here, but they suffice for the purposes of this paper. We should also mention that the distinction between persistent and durational features have proven useful for more than concurrency. In particular, it has proven fruitful for addressing the qualification problem [30].

[7] The representation of features that are only momentarily true has previously been addressed by for instance Lifschitz and Rabinov [86] and Thielscher [135].

[8] Recall that the occlusion operator X denotes that a feature is not subject to its default assumptions at a given time-point. Furthermore, the following definitions are preliminary; the final versions are presented in appendix A.

5.2 Syntactical additions

In addition to the reassignment operator R, we provide a new operator I which is typically (but not necessarily always) used for durational features. It is used to state that something holds over an interval.

$$I((\tau,\tau']\alpha) =_{def} X((\tau,\tau']\alpha) \wedge \forall t(\tau < t \leq \tau' \Rightarrow [t]\alpha) \tag{12}$$

This specifies that the features in α are exempt from their default behaviors and that the formula α is true at all time-points from $\tau + 1$ to τ'.

The definitions of a change formula and a balanced change formula are extended to include durational formulae of the forms $I((\tau,\tau']\alpha)$, $I([\tau,\tau']\alpha)$ and $I([\tau']\alpha)$, and with the same restrictions as for R and X formulae. Two new kinds of narrative statements are introduced: domain formulae (*dom*) that can contain *Per* and *Dur* elements and value formulae, and influence laws (*inf*), which have the same syntax as dependency laws.

5.3 An example

In the following narrative, there is a durational feature fire^* representing influences on the actual feature fire. Henceforth, we will follow the convention of representing the influences on an actual feature $\mathsf{f}(\overline{\omega})$ with $\mathsf{f}^*(\overline{\omega}, v)$, where v is a value in the domain of f.

dom1	$Per(\mathsf{fire}(x)) \wedge Dur(\mathsf{fire}^*(x,v),\mathsf{F})$	(13)
acs1	$[s,t]\mathsf{LightFire}(a,x) \Rightarrow I((s,t]\mathsf{fire}^*(x,\mathsf{T}))$	
inf1	$[s,s+3]\mathsf{fire}^*(x,\mathsf{T}) \wedge \neg\mathsf{fire}^*(x,\mathsf{F}) \Rightarrow R([s+3]\mathsf{fire}(x))$	
inf2	$[s]\mathsf{fire}^*(x,\mathsf{F}) \Rightarrow R([s]\neg\mathsf{fire}(x))$	
occ1	$[2,6]\mathsf{LightFire}(\mathsf{Bob},\mathsf{wood1})$	

Note how *acs1* does not immediately cause fire to be true. Instead, it produces an influence to make fire true, using the I operator. How influences on fire then affect fire is described in *inf1* and *inf2*. It takes 4 consecutive time-points to make fire true, while it takes just 1 time-point to make it false. The influence to make fire false always has precedence over the influence to make fire true. As a matter of fact, *inf1* and *inf2* are general enough to handle any conflict that can occur between a group of actions/dependencies that try to make fire both true and false at the same time. Thus, one can in principle add arbitrary action laws and dependency laws influencing fire without any worries that they lead to inconsistency. Of course, it might still be desirable to refine or modify the way conflicts between influences are treated as a domain is elaborated and more actions and dependencies are introduced.

In the examples to follow, we will continue to use influences in a manner that permits easy extension of narratives, although this practice leads to somewhat larger narratives. This issue is elaborated further in section 7.

6 Variations on the concurrency theme revisited

In section 3, a number of concurrent interactions that our original TAL formalism could not handle were identified. In this section, we show how these interactions can be represented in TAL-C. We should emphasize that although we present specific examples, the techniques employed in this section are applicable to frequently reoccurring classes of interactions.

6.1 Interactions from effects to conditions

Narrative (6) was an example of the effects of one action interfering with the execution of another concurrent action. While Bill was lighting a fire, Bob poured water on the wood. This type of interference can be handled by including a suitable condition in the influence law that makes the fire feature true.

The following two laws state that the fact that the wood is not dry produces an influence $\mathsf{fire}^*(x, \mathsf{F})$ to extinguish the fire (if there is one), and that the influence $\mathsf{fire}^*(x, \mathsf{T})$ for starting the fire has to be applied without interference for an extended period of time to affect the feature $\mathsf{fire}(x)$. The non-interference condition in this case is that $\mathsf{fire}^*(x, \mathsf{F})$ stays false.

$$\begin{array}{ll} \text{dep1} & [s]\neg\mathsf{dry}(x) \Rightarrow I([s]\mathsf{fire}^*(x, \mathsf{F})) \\ \text{inf1} & [s, s+3]\mathsf{fire}^*(x, \mathsf{T}) \wedge \neg\mathsf{fire}^*(x, \mathsf{F}) \wedge \mathsf{wood}(x) \Rightarrow \\ & \quad R([s+3]\mathsf{fire}(x)) \end{array} \tag{14}$$

Below is the complete modified version of narrative (6). The action laws *acs1* and *acs2* and dependency law *dep1* produce influences, and the effects that these influences have, alone and in combination, are specified in *inf1*,

inf2, *inf3* and *inf4*.

$$
\begin{array}{ll}
\text{dom1} & Per(\mathsf{fire}(x)) \wedge Dur(\mathsf{fire}^*(x,v),\mathsf{F}) \\
\text{dom2} & Per(\mathsf{dry}(x)) \wedge Dur(\mathsf{dry}^*(x,v),\mathsf{F}) \\
\text{dom3} & Per(\mathsf{wood}(x)) \\
\text{acs1} & [s,t]\mathsf{LightFire}(a,x) \Rightarrow I((s,t]\mathsf{fire}^*(x,\mathsf{T})) \\
\text{acs2} & [s,t]\mathsf{PourWater}(a,x) \Rightarrow I((s,t]\mathsf{dry}^*(x,\mathsf{F})) \\
\text{dep1} & [s]\neg\mathsf{dry}(x) \Rightarrow I([s]\mathsf{fire}^*(x,\mathsf{F})) \\
\text{inf1} & [s,s+3]\mathsf{fire}^*(x,\mathsf{T}) \wedge \neg\mathsf{fire}^*(x,\mathsf{F}) \wedge \mathsf{wood}(x) \Rightarrow \\
 & \quad R([s+3]\mathsf{fire}(x)) \\
\text{inf2} & [s]\mathsf{fire}^*(x,\mathsf{F}) \Rightarrow R([s]\neg\mathsf{fire}(x)) \\
\text{inf3} & [s,s+3]\mathsf{dry}^*(x,\mathsf{T}) \wedge \neg\mathsf{dry}^*(x,\mathsf{F}) \Rightarrow R([s+3]\mathsf{dry}(x)) \\
\text{inf4} & [s]\mathsf{dry}^*(x,\mathsf{F}) \Rightarrow R([s]\neg\mathsf{dry}(x)) \\
\text{obs1} & [0]\neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{dry}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}) \\
\text{occ1} & [2,6]\mathsf{LightFire}(\mathsf{bill},\ \mathsf{wood1}) \\
\text{occ2} & [3,5]\mathsf{PourWater}(\mathsf{bob},\ \mathsf{wood1})
\end{array}
\tag{15}
$$

The fact that the wood is not on fire at 7 can be inferred as follows (we provide a first-order proof in appendix A). Due to *occ2* and *acs2*, the condition $(3,5]\mathsf{dry}^*(\mathsf{wood1},\mathsf{F})$ holds. This condition and *inf4* yield $[4,5]\neg\mathsf{dry}(\mathsf{wood1})$, and as dry is persistent, $[6]\neg\mathsf{dry}(\mathsf{wood1})$ and $[7]\neg\mathsf{dry}(\mathsf{wood1})$. *Dep1* then yields $[7]\mathsf{fire}^*(\mathsf{wood1},\mathsf{F})$. Finally, *inf2* gives $[7]\neg\mathsf{fire}(\mathsf{wood1})$. Notice that although $[3,6]\mathsf{fire}^*(\mathsf{wood1},\mathsf{T})$ holds, the condition $[s,s+3](\mathsf{fire}^*(\mathsf{wood1},\mathsf{T}) \wedge \neg\mathsf{fire}^*(\mathsf{wood1},\mathsf{F}) \wedge \mathsf{wood}(\mathsf{wood1}))$ does not hold for any $s \leq 3$, and this the only condition (in *inf1*) under which $\mathsf{fire}(\mathsf{wood1})$ can become true.

The case when an effect of one action enables the effect of another action can also be handled with conditional influence laws. For instance, the following influence law states that opening a door requires initially keeping the latch open (the example is originally due to Allen [6]).

$$
\text{inf1} \quad [t]\mathsf{latch\text{-}open} \wedge [t,t+5]\mathsf{open}^*(\mathsf{T}) \Rightarrow R([t+5]\mathsf{open}) \tag{16}
$$

A variation of enablement is when the concurrent execution of two or more actions may mutually enable a common effect that none of them could have in isolation. This phenomenon is referred to as synergistic effects. It can also be the case that the concurrent execution of several actions may prevent effects that each of the actions would have in isolation. In TAL-C, this can be achieved with the use of dependency laws. One example which contains both synergistic enablement and prevention is the scenario with a soup bowl standing on a table (the version presented here is an elaboration of the original scenario, which is due to Gelfond, Lifschitz and Rabinov [44]). The

table has four sides: l for left, r for right, f for front and b for back. The variables a, x and s represent the agent, the table and a side of the table, respectively. The table can be lifted at any side (*acs1*). The actual feature $\mathsf{lift\text{-}s}(x, s)$ represents that the table x is lifted on side s. If the table is lifted at two opposite sides, then it is lifted from the ground (*dep1*), but if is not lifted at opposite sides, then it is tilted (*dep2*). If there is a soup bowl on the table and the table is tilted, then the soup is spilled (*dep3*). The relation opp, defined in *dom6*, specifies when two sides are opposite.

$$
\begin{array}{ll}
\text{dom1} & Dur(\mathsf{lift\text{-}s}(x,s),\mathsf{F}) \wedge Dur(\mathsf{lift\text{-}s}^*(x,s,v),\mathsf{F}) \\
\text{dom2} & Dur(\mathsf{tilted}(x),\mathsf{F}) \wedge Dur(\mathsf{tilted}^*(x,v),\mathsf{F}) \\
\text{dom3} & Dur(\mathsf{lifted}(x),\mathsf{F}) \wedge Dur(\mathsf{lifted}^*(x,v),\mathsf{F}) \\
\text{dom4} & Per(\mathsf{spilled}(x)) \wedge Dur(\mathsf{spilled}^*(x,v),\mathsf{F}) \\
\text{dom5} & Per(\mathsf{soup}(x)) \wedge Per(\mathsf{table}(x)) \wedge Per(\mathsf{on}(x,y)) \\
\text{dom6} & \mathsf{opp}(s,s') \equiv \\
 & [\langle s,s'\rangle = \langle \mathsf{l},\mathsf{r}\rangle \vee \langle s,s'\rangle = \langle \mathsf{r},\mathsf{l}\rangle \vee \langle s,s'\rangle = \langle \mathsf{f},\mathsf{b}\rangle \vee \langle s,s'\rangle = \langle \mathsf{b},\mathsf{f}\rangle] \\
\text{acs1} & [t_1,t_2]\mathsf{Lift}(a,x,s) \Rightarrow I((t_1,t_2]\mathsf{lift\text{-}s}^*(x,s,\mathsf{T})) \\
\text{dep1} & [t](\mathsf{table}(x) \wedge \exists s_1,s_2[\mathsf{lift\text{-}s}(x,s_1) \wedge \mathsf{lift\text{-}s}(x,s_2) \wedge \\
 & \quad \mathsf{opp}(s_1,s_2)]) \Rightarrow I([t]\mathsf{lifted}^*(x,\mathsf{T})) \\
\text{dep2} & [t](\mathsf{table}(x) \wedge \mathsf{lift\text{-}s}(x,s_1) \wedge \neg\exists s_2,s_3[\mathsf{lift\text{-}s}(x,s_2) \wedge \\
 & \quad \mathsf{lift\text{-}s}(x,s_3) \wedge \mathsf{opp}(s_2,s_3)]) \Rightarrow I([t]\mathsf{tilted}^*(x,\mathsf{T})) \\
\text{dep3} & [t]\mathsf{tilted}(y) \wedge \mathsf{on}(x,y) \wedge \mathsf{soup}(x) \Rightarrow R([t+1]\mathsf{spilled}^*(x,\mathsf{T})) \\
\text{inf1} & [t]\mathsf{lift\text{-}s}^*(x,s,\mathsf{T}) \Rightarrow I([t]\mathsf{lift\text{-}s}(x,s)) \\
\text{inf2} & [t]\mathsf{tilted}^*(x,\mathsf{T}) \Rightarrow I([t]\mathsf{tilted}(x)) \\
\text{inf3} & [t]\mathsf{spilled}^*(x,\mathsf{T}) \Rightarrow I([t]\mathsf{spilled}(x)) \\
\text{inf4} & [t]\mathsf{lifted}^*(x,\mathsf{T}) \Rightarrow I([t]\mathsf{lifted}(x)) \\
\text{obs1} & [0]\mathsf{table(t1)} \wedge \mathsf{soup(s1)} \wedge \mathsf{on(s1,t1)} \\
\text{occ1} & [3,6]\mathsf{Lift(bill,t1,l)} \\
\text{occ2} & [3,6]\mathsf{Lift(bob,t1,r)}
\end{array}
\tag{17}
$$

In this narrative, one can infer from *occ1*, *occ2*, *infl* and *inf2* that $\mathsf{lift\text{-}s(t1,l)} \wedge \mathsf{lift\text{-}s(t1,r)}$ holds from 4 to 6. Thus, the condition

$$\exists s_1,s_2[\mathsf{lift\text{-}s}(\mathsf{t1},s_1) \wedge \mathsf{lift\text{-}s}(\mathsf{t1},s_2) \wedge \mathsf{opp}(s_1,s_2)]$$

is satisfied in this time interval, enabling the effect lifted(t1) from *dep1* and *inf4*, while preventing the effect tilted(t1) according to *dep2* and *inf2*.

The table lifting scenario encodes several other potential interactions between lifting actions. If the two lifting actions are not synchronized, the table is tilted and the soup is spilled. For instance, if *occ2* is altered to

$[4,7]\mathsf{Lift(bob,t1,r)}$, then one can infer

$$[4](\mathsf{table(t1)} \wedge \mathsf{lift\text{-}s(t1,\ l)} \wedge \\ \neg\exists s_2, s_3[\mathsf{lift\text{-}s}(\mathsf{t1}, s_2) \wedge \mathsf{lift\text{-}s}(\mathsf{t1}, s_3) \wedge \mathsf{opp}(s_2, s_3)]\ ).$$

According to *dep2* and *inf2*, this produces the effect $[4]\mathsf{tilted(t1)}$, and the soup is spilled. Also notice that if the table is lifted from three sides, (for instance, add *occ3* $[4,7]\mathsf{Lift(ben,t1,f)}$ to the narrative) then the condition

$$\exists s_1, s_2[\mathsf{lift\text{-}s}(\mathsf{t1}, s_1) \wedge \mathsf{lift\text{-}s}(\mathsf{t1}, s_2) \wedge \mathsf{opp}(s_1, s_2)]$$

is still satisfied, which implies that the table is lifted and not tilted. However, if the only occurrences are *occ1* and *occ3*, then that condition does not hold and the table is tilted and the soup is spilled.

Notice that it is possible to write influence laws that determine directly from the $\mathsf{lift\text{-}s}^*$ influence whether the table is lifted or the soup is spilled. Anyhow, we have preferred to explicitly represent the causal chain from lifting to spilling and tilting, as it makes it easier to extend the scenario to include actions that for instance counter-act the lifting by pressing down a side of the table or that stabilize the bowl.

6.2 Interactions between effects

The previous subsection addressed interactions between effects and conditions of actions and dependencies. As observed in section 3, another problem is when two or more actions or dependencies are affecting the same feature. Such combinations of effects can be conflicting or cumulative.

Conflicting effects

Returning to the fire lighting narrative (15), it can be observed that the use of influence laws in that narrative also solves the problem of conflicting effects that was observed in (7) in section 3. There were two influence laws in (15) that determined the result of conflicting influences on the fire feature:

$$\begin{array}{ll} \text{inf1} & [s, s+3]\mathsf{fire}^*(x, \mathsf{T}) \wedge \neg\mathsf{fire}^*(x, \mathsf{F}) \wedge \mathsf{wood}(x) \Rightarrow \\ & \quad R([s+3]\mathsf{fire}(x)) \\ \text{inf2} & [s]\mathsf{fire}^*(x, \mathsf{F}) \Rightarrow R([s]\neg\mathsf{fire}(x)) \end{array} \tag{18}$$

Now assume that the occurrences in (15) are modified as follows.

$$\begin{array}{ll} \text{occ1} & [2,6]\mathsf{LightFire(bill,\ wood1)} \\ \text{occ2} & [4,6]\mathsf{PourWater(bob,\ wood1)} \end{array} \tag{19}$$

In this case, we can infer $(2,6]\mathsf{fire}^*(\mathsf{wood1},\mathsf{T})$ which normally results in $[6]\mathsf{fire}(\mathsf{wood1})$ (*infl*). One can also infer $[6]\neg\mathsf{dry}(\mathsf{wood1})$ and $[6]\mathsf{fire}^*(\mathsf{wood1},\mathsf{F})$, which normally has the effect $[6]\neg\mathsf{fire}(\mathsf{wood1})$ (*inf2*). The conflict between these two influences is resolved in *infl* and *inf2*, to the advantage of the latter.

The fire lighting narrative illustrates a conflict involving just two opposite influences. However, there might also be conflicts involving arbitrarily large number of influences. For instance, consider the following narrative where several agents try to pick up the same object. The feature $\mathsf{pos}(x)$ represents the position of an object x, and its value domain of positions includes both locations (e.g. floor) and agents (e.g. bill, bob). An object can only have one position, so when more than one agent is trying to take the object, then there is a conflict. This conflict is resolved in *infl* in the narrative.

$$\begin{array}{ll}
\text{dom1} & Dur(\mathsf{pos}^*(x,a),\mathsf{F}) \wedge Per(\mathsf{pos}(x)) \\
\text{acs1} & [s,t]\mathsf{Pickup}(a,x) \Rightarrow I((s,t]\mathsf{pos}^*(x,a)) \\
\text{infl} & [t]\mathsf{pos}(x)\hat{=}\mathsf{floor} \wedge \exists p[\,[t+1]\mathsf{pos}^*(x,p)\,] \Rightarrow \\
 & \quad \exists p[\,[t+1]\mathsf{pos}^*(x,p) \wedge R([t+1]\mathsf{pos}(x)\hat{=}p)\,] \\
\text{obs1} & [0]\mathsf{pos}(\mathsf{wallet})\hat{=}\mathsf{floor} \\
\text{occ1} & [2,3]\mathsf{Pickup}(\mathsf{bill},\mathsf{wallet}) \\
\text{occ2} & [2,3]\mathsf{Pickup}(\mathsf{bob},\mathsf{wallet})
\end{array} \tag{20}$$

Infl states that "if the object x is on the floor and at least one agent is trying to take x then one of the agents who are trying to take x will actually have x". The result in this case is nondeterministic, and this is perhaps the best way to treat conflicts when one lacks detailed information of what the actual result would be. Notice that the consequent of *infl* only changes the value of $\mathsf{pos}(x)$, and not the value of the influence $\mathsf{pos}^*(x,p)$. The $\mathsf{pos}^*(x,p)$ component of the consequent is in effect a filter on what values p that $\mathsf{pos}(x)$ can be reassigned to.

It is equally possible to state that no effect occurs in the case of conflict:

$$\begin{array}{ll}
\text{infl} & ([t]\mathsf{pos}(x)\hat{=}\mathsf{floor} \wedge [t+1]\mathsf{pos}^*(x,p) \wedge \\
 & \quad \neg\exists p'[\,[t+1]\mathsf{pos}^*(x,p') \wedge p \neq p'\,]) \Rightarrow R([t+1]\mathsf{pos}(x)\hat{=}p)
\end{array} \tag{21}$$

Finally, some values might be preferred to other values. For instance, we can enhance the picking-up narrative by giving preference to stronger agents, as follows. The relation stronger encodes a partial ordering on agents based on

their relative strength.

$$\begin{aligned}\text{infl}\quad & [t]\mathsf{pos}(x)\hat{=}\mathsf{floor} \wedge \exists p[\,[t+1]\mathsf{pos}^*(x,p)\,] \Rightarrow \\ & \exists p[\,[t+1]\mathsf{pos}^*(x,p) \wedge \\ & \quad \neg\exists p'[\,[t+1]\mathsf{pos}^*(x,p') \wedge \mathsf{stronger}(p',p)\,] \wedge \\ & \quad R([t+1]\mathsf{pos}(x)\hat{=}p)\,]\end{aligned} \tag{22}$$

Cumulative effects

Another common phenomenon besides conflicting influences is when influences signify some relative change, and therefore multiple influences can be combined in a cumulative way. We have already mentioned that in mechanics, multiple forces on an object can be combined using vector addition, and the vectorial sum determines changes in the object's speed and position.

Here, we present another example, which involves a box from which agents can take coins. In order to specify cumulative effects, we need to introduce a minimal portion of set theory, including set membership (*in*), the empty set (*empty*) and subtraction of one element from a set (*remove*). A set theory that is sufficient for our purpose is obtained using the following two axioms. The sets contain only features of a specific feature sort, so the axioms have to be restated for different sorts. The variable σ represents sets.

$$\forall\sigma, f, f'[in(f, remove(f', \sigma)) \equiv (in(f,\sigma) \wedge f \neq f')] \tag{23}$$

$$\forall\sigma[empty(\sigma) \equiv \forall f[\neg in(f,\sigma)]\,] \tag{24}$$

Furthermore, we provide the following definition of the sum of feature values over a set of features.

$$\begin{aligned}&\forall t, \sigma, f, m, n[in(f,\sigma) \wedge sum(t, remove(\sigma, f)) = m \wedge [t]f\hat{=}n \Rightarrow \\ &\qquad sum(t,\sigma) = (m+n)] \\ &\forall t, \sigma[empty(\sigma) \Rightarrow sum(t,\sigma) = 0]\end{aligned} \tag{25}$$

Now we can introduce an influence $\mathsf{coins}^-(a,c)$ with a value domain of natural numbers and default value 0 to represent that an agent a is taking a coin from a container c. In addition, we define the special function $\mathsf{Coins}^-(t,c)$ which for a given c represents the set of all $\mathsf{coins}^-(a,c)$ with a nonzero value at time-point t.

$$in(\mathsf{coins}^-(a,c'), \mathsf{Coins}^-(t,c)) \equiv (c = c' \wedge [t]\neg\mathsf{coins}^-(a,c)\hat{=}0) \tag{26}$$

This definition establishes the existence of a set which contains all the non-zero features of the relevant type. The definition of *remove* above then establishes the existence of all subsets of this set, which is sufficient for determining the sum of all features in the set.

The narrative (8) is modified as follows.

$$
\begin{array}{ll}
\text{dom1} & Dur(\mathsf{coins}^-(a,c),0) \wedge Per(\mathsf{coins}(c)) \\
\text{acs1} & [s,t]\mathsf{TakeCoin}(a,c) \Rightarrow I([t]\mathsf{coins}^-(a,c)\hat{=}1) \\
\text{inf1} & ([t]\mathsf{coins}(c)\hat{=}(m+n)\wedge \\
 & \quad sum(t{+}1,\mathsf{Coins}^-(t+1,c)) = n) \Rightarrow \\
 & \quad R([t+1]\mathsf{coins}(c)\hat{=}m) \\
\text{obs1} & [0]\mathsf{coins}(\mathsf{box1})\hat{=}2 \\
\text{occ1} & [2,3]\mathsf{TakeCoin}(\mathsf{bill},\mathsf{box1}) \\
\text{occ2} & [2,3]\mathsf{TakeCoin}(\mathsf{bob},\mathsf{box1})
\end{array}
\tag{27}
$$

The cumulative behavior of the coins feature is encoded in *inf1*, which adds together the values of all non-zero $\mathsf{coins}^-(a,c)$ for a specific c and adds this sum to $\mathsf{coins}(c)$. From this narrative, one can infer that the set of negative influences on box1, $\mathsf{Coins}^-(3,\mathsf{box1}) = \{\mathsf{coins}^-(\mathsf{bill},\mathsf{box1}), \mathsf{coins}^-(\mathsf{bob},\mathsf{box1})\}$. As $[3]\mathsf{coins}^-(\mathsf{bill},\mathsf{box1})\hat{=}1$ and $[3]\mathsf{coins}^-(\mathsf{bob},\mathsf{box1})\hat{=}1$ we get $sum(\mathsf{Coins}^-(3,\mathsf{box1})) = 2$ and $[3]\mathsf{coins}(\mathsf{box1})\hat{=}0$. Obviously, the narrative can be enhanced. For example, more than one coin can be taken by the same agent, coins can be added to the box, and so on.

6.3 Interacting conditions

Finally, there is the problem when the conditions of two or more actions interact. A special case of this is when an agent has a resource that can only be used for one action at a time. For instance, people generally have only two hands, and thus cannot simultaneously perform two actions that each requires the use of both hands, like lighting a fire. A strategy for representing resources is to introduce a feature representing what action the resource is actually used for. In the following narrative, the feature $\mathsf{uses\text{-}hands}(x)$ fulfills this function. There is a value sort of action tokens that duplicates the action sort (e.g. the value $\mathsf{light\text{-}fire}(a,x)$ corresponds to the action $\mathsf{LightFire}(a,x)$). The feature $\mathsf{uses\text{-}hands}(x)$ has action tokens as domain, and the letter e is used for action token variables. The action token noop stands for "no operation".

The use of a resource for an action is divided into two steps. First, the action claims the resource. *Acs1* in the narrative below states that the action of lighting a fire needs the "hands" resource. This need is represented with

the influence $\mathsf{uses\text{-}hands}^*(a, e)$. Second, the resource is actually used and the action produces some effect. *Dep1* below states that if the hands are used for lighting a fire, then this will produce a fire influence.

$$
\begin{array}{ll}
\text{dom1} & Per(\mathsf{fire}(x)) \wedge Dur(\mathsf{fire}^*(x, v), \mathsf{F}) \\
\text{dom2} & Per(\mathsf{wood}(x)) \\
\text{dom3} & Dur(\mathsf{uses\text{-}hands}(a), \mathsf{noop}) \wedge Dur(\mathsf{uses\text{-}hands}^*(a, e), \mathsf{F}) \\
\text{acs1} & [s, t]\mathsf{LightFire}(a, x) \Rightarrow \\
 & \quad I((s, t]\mathsf{uses\text{-}hands}^*(a,\ \mathsf{light\text{-}fire}(a, x))) \\
\text{dep1} & [t]\mathsf{uses\text{-}hands}(a) \hat{=} \mathsf{light\text{-}fire}(a, x) \Rightarrow I([t]\mathsf{fire}^*(x, \mathsf{T})) \\
\text{inf1} & \exists e[\,[t]\mathsf{uses\text{-}hands}^*(a, e)\,] \Rightarrow \\
 & \quad \exists e[\,[t]\mathsf{uses\text{-}hands}^*(a, e) \wedge I([t]\mathsf{uses\text{-}hands}(a) \hat{=} e)\,] \\
\text{inf2} & [s, s+3]\mathsf{fire}^*(x, \mathsf{T}) \wedge \neg\mathsf{fire}^*(x, \mathsf{F}) \wedge \mathsf{wood}(x) \Rightarrow \\
 & \quad R([s+3]\mathsf{fire}(x)) \\
\text{inf3} & [s]\mathsf{fire}^*(x, \mathsf{F}) \Rightarrow R([s]\neg\mathsf{fire}(x)) \\
\text{obs1} & [0]\mathsf{dry}(\mathsf{wood1}) \wedge \neg\mathsf{fire}(\mathsf{wood1}) \wedge \mathsf{wood}(\mathsf{wood1}) \\
\text{occ1} & [2, 6]\mathsf{LightFire}(\mathsf{bill}, \mathsf{wood1}) \\
\text{occ2} & [2, 6]\mathsf{LightFire}(\mathsf{bill}, \mathsf{wood2})
\end{array}
\tag{28}
$$

The distribution of resources is encoded in *infl*, which states that if at least one action e needs the "hands" resource then some action that needs that resource will have it (compare to *infl* in (20)). If two or more actions need a resource, then only one of them will have it, and the resource can randomly alter between competing actions. In this example, the value of $\mathsf{uses\text{-}hands}(a)$ alternates randomly between $\mathsf{light\text{-}fire}(\mathsf{bill}, \mathsf{wood1})$ and $\mathsf{light\text{-}fire}(\mathsf{bill}, \mathsf{wood2})$ from 3 to 6, with the result that both actions fail or only one of them succeeds.

More sophisticated forms of resources than the binary resource above are also possible. For instance, one can utilize the techniques presented in connection with cumulative effects to deal with quantitative and sharable resources, and resources can be renewable or consumable.

6.4 Special vs. general influences

In all examples so far, each feature type[9] has its own set of influences and influence laws. While offering a high level of freedom in handling concurrent interactions, this approach also requires declaring influences for each feature type and writing down a large amount of influence laws. Of course, if one desires a flexible and non-stereotypical treatment of interactions, this

[9] We consider all features with the same feature symbol, e.g. fire, to define a type. Several feature types with the same value domain might be of the same sort.

is hard to avoid. But TAL-C is also capable of a more uniform and compact treatment of interactions. Instead of declaring separate influences of each actual feature type (e.g. dry^* for dry), one can group together features with the same value domain and the same behavior and let them be of the same feature sort $\mathcal{F}_i$. Next, one introduces a function $* : \mathcal{F}_i \times dom(\mathcal{F}_i) \rightarrow \mathcal{F}_j$ that for a given feature and value represents an influence on the feature to change according to the value (e.g. $*(\mathsf{dry}(\mathsf{wood1}),\mathsf{F})$). Thereby, it is possible to specify influence laws that apply to all feature types that are of the sort $\mathcal{F}_i$. The following is an example of a general influence law where $dom(\mathcal{F}_i)$ is the boolean value sort. The variable f (implicitly universally quantified) is of sort $\mathcal{F}_i$. The influence law handles conflicts by making the outcome nondeterministic.

$$\begin{array}{ll} \text{dom} & Dur(*(f,v),\mathsf{F}) \\ \text{inf} & ([t]*(f,\mathsf{T}) \wedge \neg*(f,\mathsf{F}) \Rightarrow R([t]f)) \wedge \\ & ([t]\neg*(f,\mathsf{T}) \wedge *(f,\mathsf{F}) \Rightarrow R([t]\neg f)) \wedge \\ & ([t]*(f,\mathsf{T}) \wedge *(f,\mathsf{F}) \Rightarrow X([t]f)) \end{array} \tag{29}$$

This approach yields a number of influence laws that is proportional to the number of feature sorts, instead of the number of feature types. It can be particularly useful for scenarios with a large number of feature types that exhibit relatively uniform behaviors. In addition, the fact that influence laws are more general and applies to sorts rather than to specific feature types implies a higher degree of reusability.

7 Working with TAL-C narratives

When encoding a narrative in TAL-C, a bottom-up approach involving the four following levels can be used. (1) Identify relevant features of the world and their value domains. (2) For each feature, determine its normal (non-influenced) behavior, its potential influences and how these affect the feature alone and in combination. (3) Identify actions and dependencies in the world and how these influence features. (4) Determine what holds and occurs in the world, and what individuals there are.

Often, it is not possible to work strictly sequentially from level 1 to level 4. The elaboration of a complex narrative is an iterative and incremental process, where the four levels above are intertwined and decisions made earlier can be reconsidered. Therefore, to estimate how demanding this elaboration process would be in TAL-C, it is relevant to analyze what implications additions or modifications at different levels would have on an

existing narrative as a whole. Although there are some initial studies on the subject [95], there exist no systematic methods for performing this kind of estimate. Therefore, the following observations are based mainly on practical experience and commonsense.

To provide a background for the discussion to follow, we need to make some additional assumptions about the form of a narrative. We should emphasize that these assumptions represent good conventions that have been followed in this paper, but they are not formally part of the TAL-C definition. An action law for an action A has the following form, where $\Lambda_i^A(\overline{x})$ contains only actual features and $\Delta_i^A(\overline{x})$ contains only influences:

$$\text{acs} \quad [t,t']\mathsf{A}(\overline{x}) \Rightarrow \bigwedge_i(\Lambda_i^A(\overline{x}) \Rightarrow \Delta_i^A(\overline{x})) \tag{30}$$

A dependency law has the following form, with the corresponding restrictions on $\Lambda_k(\overline{x})$ and $\Delta_k(\overline{x})$.

$$\text{dep} \quad \Lambda_k(\overline{x}) \Rightarrow \Delta_k(\overline{x}) \tag{31}$$

A domain statement for a specific feature has one of the two following forms:

$$\begin{array}{ll} \text{dom} & Per(\mathsf{f}(\overline{x})) \\ \text{dom} & Dur(\mathsf{f}(\overline{x}), \mathbf{v}) \end{array} \tag{32}$$

Finally, each feature type f has a number of influence laws of the form

$$\text{inf}_i \quad \Delta_i^f(\overline{x}) \Rightarrow \Lambda_i^f(\overline{x}) \tag{33}$$

where the consequent $\Lambda_i^f(\overline{x})$ contains references to no other actual feature but $\mathsf{f}(\overline{x})$ and this feature occurs only inside reassignment, interval and occlude formulae. $\Lambda_i^f(\overline{x})$ may also contain influences that belong to $\mathsf{f}(\overline{x})$, but then only inside static subformulae. The antecedent $\Delta_i^f(\overline{x})$ contains only influences that belong to $\mathsf{f}(\overline{x})$ (e.g. $\mathsf{f}^*(\overline{x}, v)$) and actual features. Each group of influence laws represents a module that describes the behavior of a specific feature f together with the *dom* statement for that feature, and any conflicts or other interactions are handled locally within that module. Finally, we assume that each influence only belongs to one actual feature.

Given the assumptions above, we can draw the following conclusions about the impact an addition/modification will have on a narrative.

Adding a new feature (level 1 according to the enumeration above), does in itself not affect anything else. It is obviously followed by adding new influences and influence laws (level 2) and sometimes also adding/modifying actions and dependencies (level 3). These operations are discussed below.

Adding or altering the default behavior of a feature (level 2) is local to the domain statement specifying the default behavior in question. Adding a new type of influence for a feature implies altering the influence laws for the feature in question, but does not affect the influence laws for other features. If care is taken in the choice of influences, additions of influences should seldom occur, and the types of influences for a given feature should remain more or less constant. As a parallel, the physical property of an object's speed can be influenced by a large number of imaginable actions and conditions. Yet, one single type of influence ("force") suffices to determine changes in speed. Further, altering the interactions between influences for a particular feature is local to the influence laws (i.e. the module) of that feature, but does not affect the default behavior or any action or dependency laws.

Modifying or adding an action law or dependency law (level 3) does not affect any other existing action or dependency laws as any interactions are delegated to the influence laws, nor does it affect existing influence laws. The exception is of course when the new action or dependency law requires the introduction of a new type of influence for a particular feature, in which case the influence laws of that feature have to be extended. Finally, adding action occurrences and observations (level 4) does not affect anything at the preceding levels.

In summary, additions or modifications are in general local operations which preserve modularity in TAL-C. Different features can be described in isolation, and given a set of features and associated influences, different actions and dependencies can be described in isolation. This property is mainly due to two features of the logic, namely that interactions between actions and dependencies are channeled through influences, and further that the respective sets of influences of different features are disjoint, and therefore the behaviors of features can be specified in normal behavior and influence laws that are independent of those of other features. It is encouraging to achieve this level of modularity, considering the fact that we are addressing complicated causal dependencies and concurrent interactions.

8 Other work on concurrency

Hendrix's work [51] is an early attempt to represent continuous and simultaneous processes, using a STRIPS-like [39] language. Unlike STRIPS, Hendrix's formalism involves notions of explicit time, duration, and simultaneous and extraneous activity. A process has preconditions and continuation condi-

tions that determine when the process can be initiated and for how long it goes on. Hendrix distinguishes between instantaneous effects at the initiation and termination moments of the process and gradual effects that occur while the process is going on. However, there are no means for determining what happens when more than one process affects the same feature ("parameter" in Hendrix's terminology). More sophisticated representations of physical processes were later developed in qualitative reasoning, where Forbus [42] has already been mentioned.

In the work of Georgeff [45] there are world states that are linked together in histories. In a world state, features can hold and one or more events (actions) can occur. Specific features can explicitly be declared to be independent of specific events, and a persistence axiom, similar to the nochange axiom in TAL, states that if a feature p is independent of all events in a state, then it will not have changed in the next state. Georgeff then introduces the concept of *correctness conditions*. If the correctness condition p of an event e is independent of another event e', then e' will not interfere with (prevent) e. Thus, Georgeff's formalism can define when two events (actions) can and cannot occur simultaneously, and what the result is when two independent events are executed simultaneously. A limitation is the lack of an explicit notion of duration, so events cannot overlap partially. Georgeff also considers processes, which essentially are related groups of events with limited interaction with events outside the process.

Structural relations between events is the central theme in work by Lansky [77]. In GEM, there is an explicit representation of event location; events can belong to elements, which are loci of forced sequential activity, and which in turn can belong to groups. In essence, groups represent boundaries of causal access. Events inside a group can only interact with external events via specific ports (causal holes). Thereby, the possible concurrent interactions between events can be restricted.

Pelavin's work on a logic for planning with simultaneous actions with duration [109] is based on Allen's interval temporal logic [6], in which properties and actions are associated with intervals of time. Pelavin's formalism has quite a complex non-standard semantics, where the central entities are world histories. A closeness function defines how the addition of actions to a world history results in new world histories. On the syntactic level, there are modal operators on world histories: $IFTRIED(pi, P)$ denoting that the condition P would hold if the actions in pi are executed, and $INEV(i, P)$ stating that the condition P inevitably holds at time i (is independent of anything happening after i). These operators can be used for quite sophisticated descriptions of actions, including interference where one action

prevents another, and cumulative effects. However, what does not change due to an action has to be explicitly encoded, and there is no concept of dependency laws.

Thielscher [135] presents a theory of dynamic systems, where state transitions can occur naturally in addition to being caused by actions. Fluents are divided into two sets. There are *persistent fluents*, which are subject to inertia and only change when directly influenced, and there are *momentary fluents* that become false if nothing affects them. A subset of the momentary fluents are the *action fluents*. Causal laws are specified in a STRIPS-style manner, with a precondition, a set of persistent fluents to become true, a set of persistent fluents to become false and a set of momentary fluents to become true in the following state. Thielscher addresses some aspects of concurrency, but a versatile way of handling time is lacking. Durations and delays cannot be easily modeled, due to the STRIPS-style operational nature of the language. The paper also discusses one type of concurrent conflicts the formalism can handle, but no general way to handle other types of concurrent conflicts are mentioned.

Ferber and Müller [38] present a theory for dynamic multi-agent environments with a distinction between influences and state. The world develops in two-step cycles: there is a set of operators (corresponding to actions and events) that for given influences and conditions on the state yield new influences; and a set of laws that for given conditions on the state and influences transform the state. The state component develops according to a persistence assumption, whereas influences are transient (like persistent respectively durational features in TAL-C). The theory is then augmented with agent behaviors, which are functions from influence sets (percepts) to influence sets (responses). A STRIPS-style operational formalism is used in the paper, but the authors explain that the general principles should apply to other types of formalisms as well.

Among the work done in situation calculus, Pinto's [112] modeling of concurrency is particularly interesting. Pinto addresses the use of resources and exploits state constraints (of a weaker kind than the dependency laws in this paper) to deal with effect interaction, although in the context of instantaneous actions. In a recent paper [113], Pinto proposes an approach where he uses natural events (in the style of Reiter [120]) to model causal dependecies, and then he uses causal dependencies to model concurrent interactions. In this approach, natural events can play a role similar to the one of influences in TAL-C, i.e. as intermediaries of change. What is particularily interesting is that Pinto motivates his approach with arguments for modularity; using causal dependencies to model concurrent interactions

on the level of fluents allow us to describe actions in isolation. Also other authors such as Gelfond, Lifschitz and Rabinov [44], Lin and Shoham [90] and Reiter [120] address concurrency in the context of situation calculus. However, most of the problems these authors consider are specific to situation calculus, including how to extend the result function to take more than one action, and how to represent action duration [44, 120] and extraneous actions [120]. The topic of concurrent interaction is only briefly addressed; for instance, Lin's and Shoham's treatment is restricted to effect cancellation in case two actions are in conflict.

Finally, we should also mention Baral and Gelfond's propositional language $\mathcal{A}_C$ [11] and its relatives by Li and Pereira [81] and Bornscheuer and Thielscher [18]. $\mathcal{A}_C$, an extension of Gelfond's and Lifschitz's language $\mathcal{A}$ [43], relies on action rules (e-propositions) of the form $\{A_1, \ldots, A_n\}$ **causes** e **if** $p1, \ldots, p_n$ to describe concurrent interactions. Rules for the same fluent with more specific action parts override less specific ones. This makes $\mathcal{A}_C$ suitable for representing synergistic and conflicting effects. On the other hand, actual cancellation of effects requires the use of explicit frame axioms, like in the soup bowl example [11] where the rule $\{lift\_left, lift\_right\}$ **causes** $\neg spilled$ **if** $\neg spilled$ overrides the rules $\{lift\_left\}$ **causes** *spilled* and $\{lift\_right\}$ **causes** *spilled*. None of [11, 81, 18] address ramification or action duration, and only Bornscheuer's and Thielscher's version addresses nondeterminism. The three languages differ mainly in the treatment of concurrent conflicts that are not resolved by any e-proposition. According to Baral and Gelfond, the entire resulting state is undefined, according to Li and Pereira, the result of the conflicting actions is undefined, and according to Bornscheuer and Thielscher, the resulting values of the conflicting features are nondeterministic.

The fact that more specific rules override less specific ones in $\mathcal{A}_C$ makes it possible to add new rules without having to modify existing ones, thereby contributing to elaboration tolerance. Comparing $\mathcal{A}_C$ to TAL-C, it is possible to specify scenarios where certain elaborations are easy to do in $\mathcal{A}_C$ but quite complicated in TAL-C, and vice versa. However, actions cannot be described in isolation in $\mathcal{A}_C$, which has to be considered a major drawback from a modularity perspective. Also recall the discussion in section 4.1.

9 Conclusions

In this paper, we have presented TAL-C, which is a logic for describing narratives that involve action concurrency and causal dependencies between

features. What distinguishes TAL-C from previous work is the combination of the following factors: (a) TAL-C has a standard first-order semantics and proof theory. (b) TAL-C has a notion of explicit time, which makes it possible to reason about the durations of actions and other interesting temporal properties and relations. (c) TAL-C is able to model a number of important phenomena related to concurrency. We should also mention that several of the examples in this paper have been tested using an implementation of TAL and TAL-C, called VTAL (available on WWW at `http://anton.ida.liu.se/vital/vital.html` as a Java applet).

Technically, TAL-C is closely related to TAL with ramification [48], although the surface language $\mathcal{L}(ND)$ has been modified and extended. In the base language $\mathcal{L}(FL)$, the same predicates are still used, with the addition of Per and Dur. Most important, the same simple circumscription policy is still applicable, which implies that we can reason about concurrent interactions in first-order logic.

The main difference between TAL and TAL-C is conceptual in nature and based on how action laws are defined and used. We have demonstrated how traditional action laws suffer from a number of problems, in particular due to the fact that preconditions in action laws refer to the state before the action is executed, and that the effects are absolute and indefeasible. The solution involving action laws with multiple actions was rejected due to lack of precision and scaling. Instead, we proposed an approach were actions produce influences instead of actual effects. The way these influences change the world, both alone and in interaction with other influences can then be specified in influence laws for individual features. TAL-C has been demonstrated on a number of nontrivial concurrency-related problems. The use of influence laws in TAL-C provides a flexible tool for describing what happens when a feature is subject to influence from several actions or dependencies simultaneously. Additions and modifications to a TAL-C narrative are local operations which preserve modularity.

An important topic for future research is to adopt TAL-C for continuous time and value domains, in order to describe worlds with piece-wise continuous dynamics (for related work on this problem, see e.g. Sandewall [123] and Shanahan [128]). Such an adaptation will probably involve a richer representation of the default behaviors of features than the distinction between persistent and durational features used in this paper. Future research also includes systematically exploring common patterns of concurrency and establishing semantics for different classes of worlds with concurrent actions, and studying how locality [77] can be exploited in a formalism such as TAL-C.

Acknowledgements

This work has been supported by the Wallenberg foundation and by the Swedish Research Council for Engineering Sciences (TFR). Many thanks to Silvia Coradeschi, Jonas Kvarnström, Patrick Doherty, Patrik Haslum and the anonymous referees for useful comments on previous versions.

A Translation from $\mathcal{L}(ND)$ to $\mathcal{L}(FL)$

This section presents the translation from the surface language $\mathcal{L}(ND)$, which is intended for describing narratives, to $\mathcal{L}(FL)$ which is used for inferences. The translation is based on [27]. $\mathcal{L}(FL)$ is a first-order language with equality consisting of the predicates $Holds_i : \mathcal{T} \times \mathcal{F}_i \times Dom(\mathcal{F}_i)$, $Occlude_i : \mathcal{T} \times \mathcal{F}_i$ (normally, the index i is omitted) and $Occurs : \mathcal{T} \times \mathcal{T} \times \mathcal{A}$; further $Per_i : \mathcal{F}_i$, $Dur_i : \mathcal{F}_i \times Dom(\mathcal{F}_i)$ and all predicates relating to the value domains and temporal domain from $\mathcal{L}(ND)$. There is an isomorfism of sorts and symbols between $\mathcal{L}(ND)$ and $\mathcal{L}(FL)$.

A.1 Translation function

Definition 12 $Tran$ is called the *translation function*, and is defined as follows (the obvious parts have been left out). All variables occurring only on the right-hand side are assumed to be previously unused variables.

$$
\begin{aligned}
Tran([\tau]f(\overline{\omega})\hat{=}v) &= Holds(\tau, f(\overline{\omega}), v) && (34)\\
Tran([\tau]\neg\alpha) &= \neg Tran([\tau]\alpha) && (35)\\
Tran([\tau]\alpha\mathcal{C}\beta) &= Tran([\tau]\alpha)\ \mathcal{C}\ Tran([\tau]\beta) && (36)\\
&\quad \text{where } \mathcal{C} \in \{\wedge, \vee, \Rightarrow, \equiv\}.\\
Tran([\tau]\mathcal{Q}v[\alpha]) &= \mathcal{Q}v[Tran([\tau]\alpha)] \text{ where } \mathcal{Q} \in \{\forall, \exists\}. && (37)\\
Tran([\tau, \tau']\alpha) &= \forall t'[\tau \leq t' \leq \tau' \Rightarrow Tran([t']\alpha)] && (38)\\
Tran((\tau, \tau']\alpha) &= \forall t'[\tau < t' \leq \tau' \Rightarrow Tran([t']\alpha)] && (39)\\
Tran(X([\tau]f(\overline{\omega}))) &= Occlude(\tau, f(\overline{\omega})) && (40)\\
Tran(X([\tau]f(\overline{\omega})\hat{=}v)) &= Occlude(\tau, f(\overline{\omega})) && (41)\\
Tran(X([\tau]\neg\alpha)) &= Tran(X([\tau]\alpha)) && (42)\\
Tran(X([\tau]\alpha\mathcal{C}\beta)) &= Tran(X([\tau]\alpha)) \wedge Tran(X([\tau]\beta)) && (43)\\
&\quad \text{where } \mathcal{C} \in \{\wedge, \vee, \Rightarrow, \equiv\}.\\
Tran(X([\tau]\mathcal{Q}v[\alpha])) &= \forall v[Tran(X([\tau]\alpha))] \text{ where } \mathcal{Q} \in \{\forall, \exists\}. && (44)
\end{aligned}
$$

$$
\begin{aligned}
Tran(X((\tau,\tau']\alpha)) &= \forall t'[\ \tau < t' \leq \tau' \Rightarrow \\
&\qquad Tran(X([t']\alpha))] && (45)\\
Tran(X([\tau,\tau']\alpha)) &= \forall t'[\ \tau \leq t' \leq \tau' \Rightarrow \\
&\qquad Tran(X([t']\alpha))] && (46)\\
Tran(R((\tau,\tau']\alpha)) &= Tran(X((\tau,\tau']\alpha)) \wedge Tran([\tau]\alpha) && (47)\\
Tran(R([\tau,\tau']\alpha)) &= Tran(X([\tau,\tau']\alpha)) \wedge Tran([\tau]\alpha) && (48)\\
Tran(R([\tau]\alpha)) &= Tran(X([\tau],\alpha)) \wedge Tran([\tau]\alpha) && (49)\\
Tran(I((\tau,\tau']\alpha)) &= Tran(X((\tau,\tau']\alpha)) \wedge \\
&\qquad Tran((\tau,\tau']\alpha) && (50)\\
Tran(I([\tau,\tau']\alpha)) &= Tran(X([\tau,\tau']\alpha)) \wedge \\
&\qquad Tran([\tau,\tau']\alpha) && (51)\\
Tran(I([\tau]\alpha)) &= Tran(X([\tau],\alpha)) \wedge Tran([\tau]\alpha) && (52)\\
Tran(C_T([\tau]\alpha)) &= \forall t'[\tau = t' + 1 \Rightarrow Tran([t']\neg\alpha)] \wedge \\
&\qquad Tran([\tau]\alpha) && (53)\\
Tran([\tau,\tau']\Phi(\overline{\omega})) &= Occurs(\tau,\tau',\Phi(\overline{\omega})) && (54)
\end{aligned}
$$

Note the translation of the X operator, in particular lines (42–44), which always occludes all features inside a reassignment or interval formula.

A.2 Circumscription

The second-order circumscription of a number of predicates $\overline{P} = P_1, \ldots, P_n$ in the theory $\Gamma(\overline{P})$ is denoted $Circ_{SO}(\Gamma(\overline{P}); \overline{P})$ (see Lifschitz [82]). Intuitively, $Circ_{SO}(\Gamma(\overline{P}); \overline{P})$ represents a (second-order) theory containing $\Gamma(\overline{P})$ and where the extensions of the predicates $\overline{P}$ are minimal.

Definition 13 Transformation of narratives from $\mathcal{L}(ND)$ to $\mathcal{L}(FL)$:

1. Let *dom*, *acs*, *dep*, *inf*, *obs* and *occ* be the sets of statements with labels *dom*, *acs*, *dep*, *obs* and *occ* respectively, completed with universal quantification for variables occurring freely.

2. Let $\Gamma_{dom} = Tran(dom)$, $\Gamma_{acs} = Tran(acs)$, $\Gamma_{dep} = Tran(dep)$, $\Gamma_{inf} = Tran(inf)$, $\Gamma_{obs} = Tran(obs)$ and $\Gamma_{occ} = Tran(occ)$.

3. Let Γ be $Circ_{SO}((\Gamma_{acs} \cup \Gamma_{dep} \cup \Gamma_{inf})(\overline{Occlude}); \overline{Occlude}) \cup \Gamma_{dom} \cup Circ_{SO}(\Gamma_{occ}(Occurs); Occurs) \cup \Gamma_{obs} \cup \Gamma_{fl} \cup \Gamma_{fnd}$. Γ is the theory that is used for proofs in TAL-C.

The set Γ_{fl} contains the $\mathcal{L}(FL)$ equivalents of (10) and (11), plus two more axioms relating to Per and Dur.

$$
\begin{array}{l}
\Gamma_{fl} = \bigcup_i \{ \\
\forall f_i, t, v_i \, [\, Dur_i(f_i, v_i) \Rightarrow \\
\qquad (\neg Occlude_i(t, f_i) \Rightarrow (Holds_i(t, f_i, v_i)))], \\
\forall f_i, t, v_i [\, Per_i(f_i) \Rightarrow (\neg Occlude_i(t+1, f_i) \Rightarrow \\
\qquad (Holds_i(t, f_i, v_i) \equiv Holds_i(t+1, f_i, v_i)))], \\
\forall f_i, t, v_i, v_i' \; Dur_i(f, v_i) \wedge Dur_i(f, v_i') \Rightarrow v_i = v_i', \\
\forall f_i [Per_i(f_i) \oplus \exists v_i \, Dur_i(f_i, v_i)] \; \}
\end{array}
\tag{55}
$$

Finally, the set Γ_{fnd} consists of foundational axioms for unique names for actions, features and values, and constraints that a feature has exactly one value at each time-point.

An important property of the circumscribed theory Γ is that although it is a second-order theory due to the second-order nature of circumscription, it can be reduced to an equivalent first-order theory, and in a very convenient form. The following is a principal account for this reduction based on [27], where the proofs in [27] are directly applicable. Due to the definition of action laws and dependency laws, occlusion can only occur on the right-hand side Δ of an implication $\Gamma \Rightarrow \Delta$. Furthermore, due the restrictions on balanced change formulae and the definition of the $Tran$ function, occlusion only occurs positive in Δ, and if it occurs in a disjunction inside Δ, then it occurs identically on both sides. Therefore, the $Occlude_i$ parts, using the law of distributivity, can be separated from the rest of Γ, resulting in $\Lambda \Rightarrow (\bigwedge_i \Delta_i^{occ}) \wedge \Delta^h$, and from there to $\Lambda \Rightarrow \Delta^h \wedge (\bigwedge_i \Lambda \Rightarrow \Delta_i^{occ})$. Thus, it can be shown that each expanded action law or dependency law is equivalent to a formula $\Lambda \Rightarrow \Delta^h \wedge (\bigwedge_i \forall t, f_i [\Lambda' \Rightarrow Occlude_i(t, f_i)])$. Now, there are two useful theorems by Lifschitz [84], the first stating that if B does not contain P, then $Circ_{SO}(\Gamma(P) \wedge B; P) \equiv Circ_{SO}(\Gamma(P); P) \wedge B$, and the second stating that if $F(\overline{x})$ does not contain any P then $Circ_{SO}(\forall \overline{x} F(\overline{x}) \Rightarrow P(\overline{x}); P) \equiv (\forall \overline{x} F(\overline{x}) \equiv P(\overline{x}))$. From these theorems and the equivalent form above follows that $Circ_{SO}((\Gamma_{acs} \cup \Gamma_{dep})(\overline{Occlude}); \overline{Occlude}) = (\bigwedge_k \Lambda_k \Rightarrow \Delta_k^h) \wedge (\bigwedge_i \forall t, f_i [(\bigvee_k \Lambda_{ik}') \equiv Occlude_i(t, f_i)]$, which is first-order.

A.3 Example

The following is the $\mathcal{L}(FL)$ translation of narrative (15). In this case, there is only one feature sort, which has a boolean value domain.

$$\begin{array}{l}\Gamma_{dom} = \{ \\ \text{dom1}\ \forall x, v[Per(\mathsf{fire}(x)) \wedge Dur(\mathsf{fire}^*(x, v), \mathsf{F})] \\ \text{dom2}\ \forall x, v[Per(\mathsf{dry}(x)) \wedge Dur(\mathsf{dry}^*(x, v), \mathsf{F})]\ \}\end{array} \tag{56}$$

$$\begin{array}{l}\Gamma_{acs+dep+inf} = \{ \\ \text{acs1}\ \forall s, t, a, x[Occurs(s, t, \mathsf{LightFire}(a, x)) \Rightarrow \\ \qquad \forall t'[s < t' \leq t \Rightarrow Holds(t', \mathsf{fire}^*(x, \mathsf{T}), \mathsf{T})] \wedge \\ \qquad \forall t'[s < t' \leq t \Rightarrow Occlude(t', \mathsf{fire}^*(a, x, \mathsf{T}))], \\ \text{acs2}\ \forall s, t, a, x[Occurs(s, t, \mathsf{PourWater}(a, x)) \Rightarrow \\ \qquad \forall t'[s < t' \leq t \Rightarrow Holds(t', \mathsf{dry}^*(x, \mathsf{F}), \mathsf{T})] \wedge \\ \qquad \forall t'[s < t' \leq t \Rightarrow Occlude(t', \mathsf{dry}^*(x, \mathsf{F}))], \\ \text{dep1}\ \forall s, x[\neg Holds(s, \mathsf{dry}(x), \mathsf{T}) \Rightarrow Holds(s, \mathsf{fire}^*(x, \mathsf{F}), \mathsf{T})] \\ \text{inf1}\ \forall s, x(\forall t'[s \leq t' \leq s + 3 \Rightarrow (Holds(t', \mathsf{fire}^*(x, \mathsf{T}), \mathsf{T}) \wedge \\ \qquad \neg Holds(t', \mathsf{fire}^*(x, \mathsf{F})), \mathsf{T}) \wedge Holds(t', \mathsf{wood}(x), \mathsf{T}))] \Rightarrow \\ \qquad Occlude(s + 3, \mathsf{fire}(x)) \wedge Holds(s + 3, \mathsf{fire}(x), \mathsf{T}), \\ \text{inf2}\ \forall s, x[Holds(s, \mathsf{fire}^*(x, \mathsf{F}), \mathsf{T}) \Rightarrow \\ \qquad (Occlude(s, \mathsf{fire}(x)) \wedge \neg Holds(s, \mathsf{fire}(x), \mathsf{T}))], \\ \text{inf3}\ \forall s, x(\forall t'[s \leq t' \leq s + 3 \Rightarrow \\ \quad (Holds(t', \mathsf{dry}^*(x, \mathsf{T}), \mathsf{T}) \wedge \neg Holds(t', \mathsf{dry}^*(x, \mathsf{F}), \mathsf{T})] \Rightarrow \\ \quad (Occlude(s + 3, \mathsf{dry}(x)) \wedge Holds(s + 3, \mathsf{dry}(x), \mathsf{T})), \\ \text{inf4}\ \forall s, x[Holds(s, \mathsf{dry}^*(x, \mathsf{F}), \mathsf{T}) \Rightarrow \\ \qquad (Occlude(s, \mathsf{dry}(x)) \wedge \neg Holds(s, \mathsf{dry}(x), \mathsf{T}))]\}\end{array} \tag{57}$$

$$\begin{array}{l}\Gamma_{occ} = \{\ \text{occ1}\ Occurs(2, 6, \mathsf{LightFire}(\mathsf{bill}, \mathsf{wood1})), \\ \qquad \text{occ2}\ Occurs(3, 5, \mathsf{PourWater}(\mathsf{bob}, \mathsf{wood1}))\}\end{array} \tag{58}$$

$$\begin{array}{l}\Gamma_{obs} = \{\text{obs1}\ Holds(0, \mathsf{dry}(\mathsf{wood1}), \mathsf{T}) \wedge \\ \qquad \neg Holds(0, \mathsf{fire}(\mathsf{wood1}), \mathsf{T}) \wedge \\ \qquad Holds(0, \mathsf{wood}(\mathsf{wood1}), \mathsf{T})\}\end{array} \tag{59}$$

Circumscribing *Occurs* in Γ_{occ} and *Occlude* in $\Gamma_{acs} \cup \Gamma_{dep} \cup \Gamma_{inf}$ yields the following exact descriptions of the two predicates, which together with the original theory and the additional components constitute Γ. Notice that the *Occlude* part specifies exactly the exceptions to the default rules for

persistent and durational features, as expressed in the two first axioms in Γ_{fl}.

$$\begin{aligned}&\forall s,t,e\ [Occurs(s,t,e) \equiv \\ &\qquad ((s=2 \wedge t=6 \wedge e=\mathsf{LightFire(bill,wood1)}) \vee \\ &\qquad (s=3 \wedge t=5 \wedge e=\mathsf{PourWater(bob,wood1)}))]\end{aligned} \tag{60}$$

$$\begin{aligned}&\forall t',f\ (Occlude(t',f) \equiv \exists s,t,a,x[\\ &(s<t' \leq t \wedge f=\mathsf{fire}^*(x,\mathsf{T}) \wedge \\ &\quad Occurs(s,t,\mathsf{LightFire}(a,x))) \vee \\ &\ (s<t' \leq t \wedge f=\mathsf{dry}^*(x,\mathsf{F}) \wedge \\ &\quad Occurs(s,t,\mathsf{PourWater}(a,x))) \vee \\ &(f=\mathsf{fire}(x) \wedge t'=s+3 \wedge \forall t'[s \leq t' \leq s+3 \Rightarrow \\ &\qquad (Holds(t',\mathsf{fire}^*(x,\mathsf{T})),\mathsf{T}) \wedge \\ &\qquad \neg Holds(t',\mathsf{fire}^*(x,\mathsf{F}),\mathsf{T}) \wedge Holds(t',\mathsf{wood}(x),\mathsf{T})]) \vee \\ &(f=\mathsf{fire}(x) \wedge t'=s \wedge Holds(s,\mathsf{fire}^*(x,\mathsf{F}),\mathsf{T})) \vee \\ &(f=\mathsf{dry}(x) \wedge t'=s+3 \wedge (\forall t'[s \leq t' \leq s+3 \Rightarrow \\ &\quad (Holds(t',\mathsf{dry}^*(x,\mathsf{T}),\mathsf{T}) \wedge \neg Holds(t',\mathsf{dry}^*(x,\mathsf{F}),\mathsf{T}))])) \vee \\ &(f=\mathsf{dry}(x) \wedge t'=s \wedge Holds(s,\mathsf{dry}^*(x,\mathsf{F}),\mathsf{T}))])\end{aligned} \tag{61}$$

The proof for $\neg Holds(7,\mathsf{fire(wood1)},\mathsf{T})$ is as follows.

1. $Occurs(3,5,\mathsf{PourWater(bob,wood1)})$ is true according to *occ2*.
2. From 1 and *acs2*, one can infer $Holds(5,\mathsf{dry}^*(\mathsf{wood1},\mathsf{F}),\mathsf{T})$.
3. From 2 and *inf4*, it follows that $\neg Holds(5,\mathsf{dry(wood1)},\mathsf{T})$ is true.
4. From (60) and (61), it follows that $\neg Occlude(t,\mathsf{dry}^*(\mathsf{wood1},\mathsf{F}))$, $\neg Occlude(t,\mathsf{dry}^*(\mathsf{wood1},\mathsf{T}))$ and consequently $\neg Occlude(t,\mathsf{dry(wood1)})$ are true for $t=6$ and $t=7$.
5. From 3, 4, $Per(\mathsf{dry(wood1)})$ (*dom2*) and (55), it follows that $\neg Holds(t,\mathsf{dry(wood1)},\mathsf{T})$ is true for $t=6$ and $t=7$.
6. From 5 and *dep1*, it follows that $Holds(7,\mathsf{fire}^*(\mathsf{wood1},\mathsf{F}),\mathsf{T})$ is true.
7. From 6 and *inf2*, it follows that $\neg Holds(7,\mathsf{fire(wood1)},\mathsf{T})$ is true.

Paper II

Delayed Effects of Actions

Lars Karlsson, Joakim Gustafsson and Patrick Doherty
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
Email: {larka,joagu,patdo}@ida.liu.se

Abstract

A fundamental property of many dynamical systems is that effects of actions can occur with some delay. In this paper, we address the representation of delayed effects in the context of reasoning about action and change. We discuss how delayed effects can be modeled both in abstract ways and as detailed processes, and we consider a range of possible interactions between the delayed effect of an action and later occurring actions, including interference and cumulative effects. The logic used is called TAL-C, and is a logic of action and change based on explicit time that supports action duration, nondeterminism, ramification and concurrency. TAL-C uses a second-order circumscription policy with a first-order reduction.

1 Introduction

A fundamental property of many dynamical environments, in particular natural ones, is that changes as a response to actions or events do not occur instantaneously, but after some duration of time. This observation is obviously of great interest in reasoning about action and change, and is usually discussed in terms of delayed effects or processes. Actually, if some change

occurs after the end of an action, then there must be some underlying process going on. A description of this process might be too complicated or not even available. By considering this change as a delayed effect of the action in question, one can sometimes abstract away from the details of the process and still be able to obtain an adequate description of its manifestations. In this paper, we show how delayed effects can be modeled in the language TAL-C [64]. TAL-C is a member of a family of logics derived from Sandewall's PMON logic [124, 26] for sequential scenarios with explicit time and actions with context-dependent and non-deterministic effects and extended duration. Dependency laws as a means for dealing with ramifications were added in [48], and are exploited for concurrency in TAL-C. Recently, the qualification problem has also been addressed [30]. There are three factors that make TAL-C a suitable instrument for dealing with delayed effects: the existence of a notion of time which is independent of actions; the possibilities to define causal dependencies outside of action descriptions; and the support for concurrent interactions. Explicit time makes it possible to state that the delayed effect occurs after some (possibly indeterminate) amount of time, and causal dependencies and concurrent interactions are crucial in describing interactions between actions. The fact that the effects of an action are not confined temporally within the duration of that action creates ample opportunities for interactions to occur: a delayed effect of an action can prevent or be prevented by the effects of a later action, or might occur simultaneously with a later action.

Delayed effects have not received much attention in the literature. Some papers on continuous change, e.g. by Sandewall [123] and Shanahan [128], address the explicit representation of continuous processes that can go on after an action has been executed, such as water filling a sink after the tap has been turned on. Shanahan also presents an example with an alarm that goes off after a fixed time interval. Further, there is work by Doherty and Gustafsson [28] about representing delays with dependency laws. However, none of the logics in these papers deal with concurrent interactions. It is the modeling of how the delayed effect of one action can interact with other actions that is the most important contribution of this article.

An example due to Gelfond, Lifschitz and Rabinov [44] of a delayed effect is a pedestrian light, which turns green 30 seconds after one presses the button at the crosswalk. They attempt to represent the delayed effect as a postcondition of a Press action. In TAL-C, the same example (somewhat extended) is encoded in the scenario description below, followed by a detailed

explanation. The numbers and t variables represent time-points.

$$
\begin{array}{ll}
\text{dom1} & \mathit{Dur}(\mathsf{pressed}, \mathsf{F}) \wedge \mathit{Per}(\mathsf{tick}) \wedge \mathit{Per}(\mathsf{color}) \\
\text{acs1} & [t, t']\mathsf{Press} \Rightarrow I((t, t']\mathsf{pressed}) \\
\text{dep1} & C_T([t]\mathsf{pressed} \wedge \neg\mathsf{tick}) \Rightarrow R([t+1]\mathsf{tick}) \\
\text{dep2} & C_T([t]\mathsf{tick}) \wedge [t, t+29]\mathsf{tick} \Rightarrow R([t+29]\mathsf{color} \hat{=} \mathsf{green}) \\
\text{dep3} & C_T([t]\mathsf{tick}) \wedge [t, t+58]\mathsf{tick} \Rightarrow R([t+59]\mathsf{color} \hat{=} \mathsf{red} \wedge \neg\mathsf{tick}) \\
\text{obs1} & [0]\mathsf{color} \hat{=} \mathsf{red} \wedge \neg\mathsf{tick} \\
\text{occ1} & [10, 14]\mathsf{Press} \\
\text{occ2} & [22, 26]\mathsf{Press}
\end{array}
\tag{1}
$$

The contents of a scenario description are given in a compact high-level macro language which is translated to a standard first-order logic theory that is used for reasoning about the scenario; section 2 describes this translation. A scenario description contains statements relating to the object and feature domains (labeled *dom*) and general action and dependency laws (*acs* and *dep*). There are also action occurrences (*occ*), all of which are assumed to be known but their timing might be partially specified, and observations of states of the world at different points in time (*obs*).

Beginning with *dom1*, a distinction is made between two types of features: persistent features whose normal behavior is not to change (this is the predominant type in the RAC literature); and durational features whose normal behavior is to have some suitable default value, but permits a non-default value under certain conditions such as during the execution of an action. In this scenario, pressed is a durational feature with default value F, and tick and color are persistent features. The pressed feature (which normally is false) will be true while the Press action is performed, as described by *acs1*. In general, the form $I((\tau, \tau']\alpha)$ that is used in *acs1* denotes that the features in α are not subject to their default behaviors and that the formula α is true at all time-points from $\tau + 1$ to τ'. The temporal argument can also be given as $[\tau, \tau']$ denoting that τ is included in the interval, and $[\tau]$, referring to a single time-point. The pressed feature serves as an *influence*, whose purpose it is convey change to other features, in this case tick. *Dep1* states that if the condition $[t]\mathsf{pressed} \wedge \neg\mathsf{tick}$ becomes true (C_T should be read as "changes to true") then tick becomes true. The form $R([\tau']\alpha)$ used in *dep1*, which also can have temporal arguments $(\tau, \tau']$ or $[\tau, \tau']$, overrides the default behavior and makes α true at τ' (that is, at the end of the interval). There is also a form $X((\tau, \tau']\alpha)$ that lets the features involved vary arbitrarily. *Dep2* states that if the ticking goes on for 30 s, then the pedestrian light becomes green, and *dep3* states that the light switches back to red and the ticking ends after an additional 20 s. Finally, there is an observation (*obs1*)

stating that the ticking is off and the light is red at time-point 0, and two occurrences of the Press action (*occ1* and *occ2*).

There is some subtlety involved with this scenario description. Pressing the button a second time (while tick is true) will not result in a second period of green. Thus, the second pressing of the button (*occ2*) is futile. Note that this represents an interaction between a delayed effect of one action (*occ1*) and a second intermediate action (*occ2*). In addition, the delayed effect itself (the light is green) has a duration.

A presentation of the technicalities of TAL-C is found in section 2 followed by further examples and a more elaborate investigation into delayed effects in section 3. Finally, section 4 concludes with a discussion.

2 The TAL-C Language

TAL-C consists of two language levels. First, there is the surface language $\mathcal{L}(ND)$ which is used in scenario descriptions such as the one above. These scenario descriptions are then translated to the standard first-order language $\mathcal{L}(FL)$, and some of the predicates in $\mathcal{L}(FL)$ are completed by means of circumscription. The framework is fully described in [27, 64].

2.1 The surface language $\mathcal{L}(ND)$

There are a number of sorts for values $\mathcal{V}_i$ (including agents and various types of objects). One of these sorts is $\mathcal{B}$ with the constants $\{\mathsf{T}, \mathsf{F}\}$. Further, there are a number of sorts for features $\mathcal{F}_i$, each one associated with a value domain $dom(\mathcal{F}_i) = \mathcal{V}_j$ for some j, and a sort for actions $\mathcal{A}$. A *value term* is a term of sort $\mathcal{V}_i$ for some i. Finally, there is a temporal sort $\mathcal{T}$ associated with a number of constants $0, 1, 2, 3, \ldots$ and $\mathsf{s}_1, \mathsf{t}_1, \ldots$ and a function $+$ and three predicates $=$, $<$ and $\leq$. $\mathcal{T}$ is assumed to be an interpreted sort, but can be axiomatized in first-order logic as a subset of Presburger arithmetic [68] (natural numbers with addition).

An *elementary fluent formula* has the form[1] $f(\overline{\omega})\hat{=}v$ where f is a feature symbol and $\overline{\omega}$, v are value terms, or $f(\overline{\omega})$ if $Dom(f) = \mathcal{B}$. A *fluent formula* is an elementary fluent formula or a combination of fluent formulae formed with the standard logical connectives and quantifiers.

Let τ, τ' be temporal terms and α a fluent formula. Then $[\tau, \tau']\alpha$ and $[\tau]\alpha$ are *fixed fluent formulae*, $C_T([\tau]\alpha)$ is a *becomes formula*, $R((\tau, \tau']\alpha)$,

[1] We use the overline as abbreviation of a sequence, when the contents of the sequence is obvious. For example, $f(\overline{x}, \overline{y})$ means $f(x_1, ..., x_n, y_1, ..., y_m)$.

$R([\tau,\tau']\alpha)$ and $R([\tau]\alpha)$ are *reassignment formulae*, $I((\tau,\tau']\alpha)$, $I([\tau,\tau']\alpha)$ and $I([\tau]\alpha)$ are *interval formulae*, and $X((\tau,\tau']\alpha)$, $X([\tau,\tau']\alpha)$ and $X([\tau]\alpha)$ are *occlusion formulae*.

A logical combination (including quantifiers) of temporal and value formulae (of the forms $\tau = \tau'$, $\tau < \tau'$, $\tau \leq \tau'$ and $\omega = \omega'$), fixed fluent formulae and/or becomes formulae is called a *static formula*. Static formulae are used in the preconditions of action and dependency laws, and in observations.

A *change formula* is a formula that has (or is rewritable to) the form, $\mathcal{Q}\overline{v}(\alpha_1 \vee ... \vee \alpha_n)$ where $\mathcal{Q}\overline{v}$ is a sequence of quantifiers with variables, and each α_i is a conjunction of static, occlusion, reassignment and interval formulae. The change formula is called *balanced* iff the following two conditions hold. (a) Whenever a feature $f(\overline{\omega})$ appears inside a reassignment, interval or occlusion formula in one of the α_i disjuncts, then it must also appear in all other α_i's inside a reassignment, interval or occlusion formula with exactly the same temporal argument. (b) Any existentially quantified variable v in the formula, whenever appearing inside a reassignment, interval or occlusion formula, only does so in the position $f(\overline{\omega})\hat{=}v$.

An *application formula* is any of the following: (a) A balanced change formula. (b) $\Lambda \Rightarrow \Delta$, where Λ is a static formula and Δ is a balanced change formula. (c) A conjunction of application formulae. (d) $\forall t[\Phi]$, where Φ is an application formula. (e) $\forall v[\Phi]$, where Φ is an application formula.

An *occurrence formula* has the form $[\tau,\tau']\Phi(\overline{\omega})$, where τ and τ' are elementary time-point expressions and $\Phi(\overline{\omega})$ is an action term.

An action law (labeled *acs*) has the form $[t,t']\Phi(\overline{x}) \Rightarrow \Psi(\overline{x},\overline{y})$ where $[t,t']\Phi(\overline{x})$ is an occurrence formula and $\Psi(\overline{x},\overline{y})$ is an application formula. A dependency law (labeled *dep*) is an application formula. An observation (labeled *obs*) is a static formula. An occurrence (labeled *occ*) is an occurrence formula $[\tau,\tau']\Phi(\overline{\omega})$ where τ, τ', $\overline{\omega}$ all are constants. A domain formula (labeled *dom*) is a conjunction of *Per* and *Dur* statements that classify features, and other domain-specific statements of choice.

2.2 Translation from $\mathcal{L}(ND)$ to $\mathcal{L}(FL)$

$\mathcal{L}(FL)$ is a many-sorted first-order language consisting of the predicates $Holds_i : \mathcal{T} \times \mathcal{F}_i \times Dom(\mathcal{F}_i)$, $Occlude_i : \mathcal{T} \times \mathcal{F}_i$, $Occurs : \mathcal{T} \times \mathcal{T} \times \mathcal{A}$, $Per_i : \mathcal{F}_i$ and $Dur_i : \mathcal{F}_i \times Dom(\mathcal{F}_i)$ and all predicates and functions relating to the value domains and temporal domain from $\mathcal{L}(ND)$. There is an isomorphism of sorts and names between $\mathcal{L}(ND)$ and $\mathcal{L}(FL)$.

$Tran$ is called the *translation function*, and the most interesting parts are as follows (for the complete version, see [64]). All variables occurring

only on the right-hand side are assumed to be previously unused variables.

$$
\begin{array}{lcl}
Tran([\tau]f(\overline{\omega})\hat{=}v) & = & Holds(\tau, f(\overline{\omega}), v) \\
Tran(X([\tau]f(\overline{\omega})\hat{=}v)) & = & Occlude(\tau, f(\overline{\omega})) \\
Tran(X([\tau]\neg\alpha)) & = & Tran(X([\tau]\alpha)) \\
Tran(X([\tau]\alpha\mathcal{C}\beta)) & = & Tran(X([\tau]\alpha)) \wedge Tran(X([\tau]\beta)) \\
 & & \text{where } \mathcal{C} \in \{\wedge, \vee, \Rightarrow, \equiv\}. \\
Tran(X([\tau]\mathcal{Q}v[\alpha])) & = & \forall v[Tran(X([\tau]\alpha))] \text{ where } \mathcal{Q} \in \{\forall, \exists\}. \\
Tran(X((\tau, \tau']\alpha)) & = & \forall t'[\tau < t' \leq \tau' \Rightarrow Tran(X([t']\alpha))] \\
Tran(X([\tau, \tau']\alpha)) & = & \forall t'[\tau \leq t' \leq \tau' \Rightarrow Tran(X([t']\alpha))] \\
Tran(R((\tau, \tau']\alpha)) & = & Tran(X((\tau, \tau'], \alpha)) \wedge Tran([\tau]\alpha) \\
Tran(I((\tau, \tau']\alpha)) & = & Tran(X((\tau, \tau']\alpha)) \wedge Tran([\tau{+}1, \tau']\alpha) \\
Tran(C_T([\tau]\alpha)) & = & \forall t'[\tau = t' + 1 \Rightarrow Tran([t']\neg\alpha)] \wedge \\
 & & Tran([\tau]\alpha) \\
Tran([\tau, \tau']\Phi(\overline{\omega})) & = & Occurs(\tau, \tau', \Phi(\overline{\omega}))
\end{array}
$$

Note how the R and I operators are defined in terms of the X operator and a fixed fluent formula. X is in turn mapped to *Occlude* and overrides the normal behaviors of the features involved, and the fixed fluent formula is mapped to *Holds* and states what values the features involved should have. Note the translation of the X operator which will always occlude all features referenced inside a reassignment or interval formula, thereby allowing them to change (e.g. $Tran(X([1]\mathsf{f} \vee \neg\mathsf{g})) = Occlude(1, \mathsf{f}) \wedge Occlude(1, \mathsf{g})$). The result of applying $Tran$ to the pedestrian light scenario is as follows.

$$
\begin{array}{ll}
\text{dom1} & Dur_1(\mathsf{pressed}, \mathsf{F}) \wedge Per_1(\mathsf{tick}) \wedge Per_2(\mathsf{color}) \\
\text{acs1} & \forall t, t'[Occurs(t, t', \mathsf{Press}) \Rightarrow \forall u[t < u \leq t' \Rightarrow \\
 & \quad Holds_1(u, \mathsf{pressed}, \mathsf{T}) \wedge Occlude_1(u, \mathsf{pressed})]] \\
\text{dep1} & \forall t[Holds_1(t, \mathsf{pressed}, \mathsf{T}) \wedge \neg Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
 & \quad \forall u[t = u + 1 \Rightarrow \neg(Holds_1(u, \mathsf{pressed}, \mathsf{T}) \wedge \\
 & \qquad \neg Holds_1(u, \mathsf{tick}, \mathsf{T}))] \Rightarrow \\
 & \quad Holds_1(t{+}1, \mathsf{tick}, \mathsf{T}) \wedge Occlude_1(t{+}1, \mathsf{tick})]] \\
\text{dep2} & \forall t[(Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
 & \quad \forall u[t = u + 1 \Rightarrow \neg Holds_1(u, \mathsf{tick}, \mathsf{T})] \wedge \\
 & \quad \forall u[t \leq u \leq t{+}29 \Rightarrow Holds_1(u, \mathsf{tick}, \mathsf{T})]) \Rightarrow \\
 & \quad (Holds_2(t{+}29, \mathsf{color}, \mathsf{green}) \wedge Occlude_2(t{+}29, \mathsf{color}))] \\
\text{dep3} & \forall t[(Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
 & \quad \forall u[t = u{+}1 \Rightarrow \neg Holds_1(u, \mathsf{tick}, \mathsf{T})] \wedge \\
 & \quad \forall u[t \leq u \leq t{+}58 \Rightarrow Holds_1(u, \mathsf{tick}, \mathsf{T})]) \Rightarrow \\
 & \quad (Holds_2(t{+}59, \mathsf{color}, \mathsf{red}) \wedge Occlude_2(t{+}59, \mathsf{color}) \wedge \\
 & \quad \neg Holds_1(t{+}59, \mathsf{tick}, \mathsf{T}) \wedge Occlude_1(t{+}59, \mathsf{tick}))]
\end{array}
\tag{2}
$$

obs1 $Holds_2(0, \mathsf{color}, \mathsf{red}) \wedge \neg Holds_1(0, \mathsf{tick}, \mathsf{T})$
occ1 $Occurs(10, 14, \mathsf{Press})$
occ2 $Occurs(22, 26, \mathsf{Press})$

The translation to $\mathcal{L}(FL)$ also involves a non-monotonic step, which essentially consists of forming the predicate completions of *Occlude* and *Occurs*, and adding a number of auxiliary axioms concerning among other things the default behaviors of persistent and durational features.

The second-order circumscription [82] of a number of predicates $\overline{P} = P_1, \ldots, P_n$ in the theory $\Gamma(\overline{P})$ is denoted

$$Circ_{SO}(\Gamma(\overline{P}); \overline{P}).$$

Intuitively, $Circ_{SO}(\Gamma(\overline{P}); \overline{P})$ represents a (second-order) theory containing $\Gamma(\overline{P})$ and where the extensions of the predicates $\overline{P}$ are minimal.

Definition 14 Transformation of scenarios from $\mathcal{L}(ND)$ to $\mathcal{L}(FL)$:

1. Let *dom*, *acs*, *dep*, *obs* and *occ* be the sets of statements with labels dom, acs, dep, obs and occ respectively, completed with universal quantification for variables occurring free.

2. Let $\Gamma_{dom} = Tran(dom)$, $\Gamma_{acs} = Tran(acs)$, $\Gamma_{dep} = Tran(dep)$, $\Gamma_{obs} = Tran(obs)$ and $\Gamma_{occ} = Tran(occ)$.

3. Let Γ be $Circ_{SO}((\Gamma_{acs} \cup \Gamma_{dep})(\overline{Occlude}); \overline{Occlude})$ $\cup \Gamma_{dom} \cup Circ_{SO}(\Gamma_{occ}(Occurs); Occurs) \cup \Gamma_{obs} \cup \Gamma_{fl}$ $\cup \Gamma_{fnd}$. Γ is the theory that is used for proofs in TAL-C.

The set Γ_{fl} contains two axioms stating that durational features that are not occluded must always have their default value, and persistent features that are not occluded must always keep their value from the previous time point. Further, durational features can have only one default value and all features are either persistent or durational ($\oplus$ is exclusive or).

$$\begin{aligned}
&\Gamma_{fl} = \textstyle\bigcup_i \{ \\
&\forall f_i, t, v_i \, [Dur_i(f_i, v_i) \Rightarrow \\
&\quad (\neg Occlude_i(t, f_i) \Rightarrow Holds_i(t, f_i, v_i))], \\
&\forall f_i, t, v_i [Per_i(f_i) \Rightarrow (\neg Occlude_i(t+1, f_i) \Rightarrow \\
&\quad (Holds_i(t, f_i, v_i) \equiv Holds_i(t+1, f_i, v_i)))], \\
&\forall f_i, t, v_i, v_i' [Dur_i(f, v_i) \wedge Dur_i(f, v_i') \Rightarrow v_i = v_i'], \\
&\forall f_i [Per_i(f_i) \oplus \exists v_i \, Dur_i(f_i, v_i)] \; \}
\end{aligned} \tag{3}$$

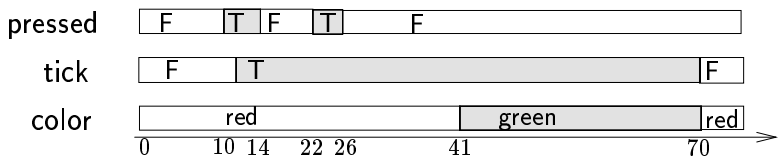

Figure 1: Model of the pedestrian scenario.

Finally, the set Γ_{fnd} consists of foundational axioms for unique names for actions, features and values, and constraints that a feature has exactly one value at each time-point.

An important property of the circumscribed theory Γ is that although it is a second-order theory due to the second-order nature of circumscription, it can be reduced to an equivalent first-order theory of the form

$$\Gamma \wedge \bigwedge_i (\forall f, t[Occlude_i(t, f) \equiv \Delta_i]) \wedge (\forall a, t, t'[Occurs(t, t', a) \equiv \Lambda])$$

(see [64, 27] for details). Thus, one can reason about TAL-C scenarios in first-order logic. The completions of the *Occurs* and *Occlude* predicates in (2) are as follows.

$$\begin{aligned}
&\forall t, t', a[Occurs(t, t', a) \equiv (\langle t, t', a\rangle = \langle 10, 14, \mathsf{Press}\rangle \vee \\
&\qquad \langle t, t', a\rangle = \langle 22, 26, \mathsf{Press}\rangle)] \wedge \\
&\forall s, f[Occlude_1(s, f) \equiv \\
&\ (\exists t, t'[Occurs(t, t', \mathsf{Press}) \wedge t < s \leq t'] \wedge f = \mathsf{pressed}) \vee \\
&\ \ \exists t[Holds_1(t, \mathsf{pressed}, \mathsf{T}) \wedge \neg Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
&\ \ \ \forall u[t = u{+}1 \Rightarrow \neg(Holds_1(u, \mathsf{pressed}, \mathsf{T}) \wedge \\
&\qquad \neg Holds_1(u, \mathsf{tick}, \mathsf{T}))] \wedge s = t + 1 \wedge f = \mathsf{tick}] \vee \\
&\ \ \exists t[f = \mathsf{tick} \wedge s = t{+}59 \wedge Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
&\ \ \ \forall u[t = u{+}1 \Rightarrow \neg Holds_1(u, \mathsf{tick}, \mathsf{T})] \wedge \\
&\ \ \ \forall u[t \leq u \leq t{+}58 \Rightarrow Holds_1(u, \mathsf{tick}, \mathsf{T})]]] \wedge \\
&\forall s, f[Occlude_2(s, f) \equiv (f = \mathsf{color} \wedge \\
&\ (\exists t[s = t{+}29 \wedge (Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
&\ \ \ \forall u[t = u{+}1 \Rightarrow \neg Holds_1(u, \mathsf{tick}, \mathsf{T})] \wedge \\
&\ \ \ \forall u[t \leq u \leq 29 \Rightarrow Holds_1(u, \mathsf{tick}, \mathsf{T})])] \vee \\
&\ \ \exists t[s = t{+}59 \wedge (Holds_1(t, \mathsf{tick}, \mathsf{T}) \wedge \\
&\ \ \ \forall u[t = u{+}1 \Rightarrow \neg Holds_1(u, \mathsf{tick}, \mathsf{T})] \wedge \\
&\ \ \ \forall u[t \leq u \leq 58 \Rightarrow Holds_1(u, \mathsf{tick}, \mathsf{T})])]))]
\end{aligned} \tag{4}$$

Figure 1 shows the single model for the pedestrian light scenario. The grey segments are derivable from (2), and represent where the features are

affected by actions and dependency laws (and hence are occluded). The white parts are the result of (3) and (4), which define the normal behaviors of the features and where these normal behaviors are applicable (these parts are not occluded).

3 Examples

In the pedestrian light scenario in the introduction, we gave an example of a delayed effect that had limited duration and that blocked another effect (i.e. the effect of the second pressing of the button). In this section, we investigate some other interesting aspects of delayed effects using some illustrative scenarios.

Interruption and varying delay time. The first of these scenarios illustrates that the delay may be of varying length and that a delayed effect might be interrupted. It involves a fire-cracker which explodes after a delay of between 10 and 15 seconds, and provides a good picture of how concurrency is handled in TAL-C. There are two features, burn and expl, which are determined by three influences: $\mathsf{burn}^*(\mathsf{T})$ and $\mathsf{burn}^*(\mathsf{F})$ make burn true and false respectively, and expl^* causes the cracker to explode. The effects of the actions and dependencies in *acs1*, *acs2*, *dep4*, *dep5* and *dep6* are directed through these influences, and *dep1*, *dep2* and *dep3* specify how the influences affect burn and expl, including when they are in conflict. Regarding delays, *dep5* states that if the fuse has been burning for at least 10 time-points, then it might explode, and *dep6* states that the cracker will explode if the fuse has been burning for 15 time-points. This scenario yields the conclusion that there will be an explosion sometimes between time-points 13 and 18.

$$
\begin{array}{ll}
\text{dom1} & \mathit{Per}(\mathsf{burn}) \wedge \mathit{Dur}(\mathsf{expl},\mathsf{F}) \wedge \mathit{Dur}(\mathsf{burn}^*(v),\mathsf{F}) \wedge \mathit{Dur}(\mathsf{expl}^*,\mathsf{F}) \qquad (5)\\
\text{acs1} & [t,t']\mathsf{Light} \Rightarrow I((t,t']\mathsf{burn}^*(\mathsf{T}))\\
\text{acs2} & [t,t']\mathsf{Extinguish} \Rightarrow I((t,t']\mathsf{burn}^*(\mathsf{F}))\\
\text{dep1} & [t](\mathsf{burn}^*(\mathsf{T}) \wedge \neg\mathsf{burn}^*(\mathsf{F})) \Rightarrow R([t]\mathsf{burn})\\
\text{dep2} & [t]\mathsf{burn}^*(\mathsf{F}) \Rightarrow R([t]\neg\mathsf{burn})\\
\text{dep3} & [t]\mathsf{expl}^* \Rightarrow I([t]\mathsf{expl})\\
\text{dep4} & [t]\mathsf{expl} \Rightarrow I([t]\mathsf{burn}^*(\mathsf{F}))\\
\text{dep5} & [t,t{+}9]\mathsf{burn} \Rightarrow X([t{+}10]\mathsf{expl}^*)\\
\text{dep6} & [t,t{+}14]\mathsf{burn} \Rightarrow I([t{+}15]\mathsf{expl}^*)\\
\text{obs1} & [0]\neg\mathsf{burn}\\
\text{occ1} & [2,3]\mathsf{Light}
\end{array}
$$

However, adding an Extinguish action will interrupt the burning, according to *acs2* and *dep2*.

$$\text{occ2} \quad [5,6]\mathsf{Extinguish} \tag{6}$$

Notice that additional action laws that relate to burn can be added incrementally by utilizing burn^*, without any need for modifying the underlying description of burn itself. This is one of the strengths of TAL-C.

Concurrent interaction. Concurrent interaction due to delayed effects can also be of a more direct nature, like in the following example with cumulative effects. In order to specify cumulative effects, we need to introduce a minimal portion of set theory: set membership (*in*), the empty set (*empty*) and removal of one element from a set (*rem*). Furthermore, we provide the following definition of the sum of feature values over a set of features. The variable σ represents sets.

$$\begin{array}{l} \forall t, \sigma, f, m, n[(in(f,\sigma) \wedge sum(t, rem(\sigma, f)) = m \wedge \\ \quad [t]f \hat{=} n) \Rightarrow sum(t,\sigma) = (m+n)] \\ \forall t, \sigma[empty(\sigma) \Rightarrow sum(t,\sigma) = 0] \end{array} \tag{7}$$

The following scenario contains a bank account, where the balance (the persistent numerical feature balance) can be altered by influences $\mathsf{bal}^+(x)$ with default value 0, where x represents the source (person, bill etc) of the influence. A special function $\mathsf{Bal}^+(t)$ represents the set of non-zero influences $\mathsf{bal}^+(x)$ at each time-point t.

$$in(f, \mathsf{Bal}^+(t)) \equiv \exists x[f = \mathsf{bal}^+(x) \wedge [t]\neg \mathsf{bal}^+(x) \hat{=} 0] \tag{8}$$

The values of the members of the set $\mathsf{Bal}^+(t)$ are added together at each time-point t, and this sum is added to the present balance. We also assume that there can be a deficit on the bank account, but that this leads to a remark from the bank. The following scenario describes how an agent (bob) sends a bill to be paid by his bank and then realizes that the balance is too low to cover this bill, and in the last moment manages to make a deposit

and avoids getting a remark.

$$
\begin{array}{ll}
\text{dom1} & \mathit{Per}(\mathsf{balance}) \wedge \mathit{Dur}(\mathsf{bal}^+(x), 0) \wedge \\
& \mathit{Per}(\mathsf{amount}(x)) \wedge \mathit{Per}(\mathsf{remark}) \\
\text{acs1} & [s,t]\mathsf{Deposit}(a,n) \Rightarrow I([t]\mathsf{bal}^+(a) \hat{=} n) \\
\text{acs2} & [s,t]\mathsf{MailBill}(a,b) \wedge [t]\mathsf{amount}(b) \hat{=} n \Rightarrow \\
& \quad I([t+50]\mathsf{bal}^+(b) \hat{=} -n) \\
\text{dep1} & [t]\mathsf{balance} \hat{=} m \wedge \mathit{sum}(t{+}1, \mathsf{Bal}^+(t{+}1)) = n \Rightarrow \\
& \quad R([t+1]\mathsf{balance} \hat{=} (m+n)) \\
\text{dep2} & C_T([t]\exists n[\mathsf{balance} \hat{=} n \wedge n < 0]) \Rightarrow R([t]\mathsf{remark}) \\
\text{obs1} & [0]\mathsf{balance} \hat{=} 50 \wedge \neg\mathsf{remark} \\
\text{obs2} & [0]\mathsf{amount}(\mathsf{bill1}) \hat{=} 150 \\
\text{occ1} & [2,3]\mathsf{MailBill}(\mathsf{bob}, \mathsf{bill1}) \\
\text{occ2} & [52,53]\mathsf{Deposit}(\mathsf{bob}, 100)
\end{array}
\tag{9}
$$

The key here is *dep1*, which summarizes the values of the non-zero $\mathsf{bal}^+(a)$ in $\mathsf{Bal}^+(t)$ and adds this sum to balance. Thus,

$$\mathsf{Bal}^+(53) = \{\mathsf{bal}^+(\mathsf{bob}), \mathsf{bal}^+(\mathsf{bill1})\}$$

where $[53]\mathsf{bal}^+(\mathsf{bob}) \hat{=} 100$ and $[53]\mathsf{bal}^+(\mathsf{bill1}) \hat{=} -150$, giving

$$\mathit{sum}(53, \mathsf{Bal}^+(53)) \hat{=} -50$$

and $[53]\mathsf{balance} \hat{=} 0$.

Explicit processes. The previous examples have shown how one can abstract away the underlying mechanics of processes. It is, however, important to point out that our approach is well suited for scenarios that model the dynamics of the world in more detail by letting features represent actual physical quantities. For instance, consider somebody filling a jug from a beer cask with a tap. Let the real-valued feature $\mathsf{flow}(x,y)$ represent a flow from x to y and let $\mathsf{Flow}^+(t,y)$ represent the set of non-zero flows into y at t and let $\mathsf{Flow}^-(t,x)$ represent the non-zero flows out of x.

$$
\begin{array}{l}
\mathit{in}(f, \mathsf{Flow}^+(t,y)) \equiv \exists x[f = \mathsf{flow}(x,y) \wedge [t]\neg\mathsf{flow}(x,y) \hat{=} 0] \\
\mathit{in}(f, \mathsf{Flow}^-(t,x)) \equiv \exists y[f = \mathsf{flow}(x,y) \wedge [t]\neg\mathsf{flow}(x,y) \hat{=} 0]
\end{array}
\tag{10}
$$

In the following scenario, *dep2* encodes a difference equation that determines the volume of water in a vessel based on the sum of flows into and out of the vessel. There are also actions for opening and closing taps (*acs1* and *acs2*), and a dependency law that states that a cask with an open tap produces a flow which is a function of the level of liquid inside the cask (*dep1*). A_1

is the bottom area of the cask, A_2 is the area of the tap hole and g is the gravitation constant. Initially, the jug is under the tap of the cask (*obs1*) and the tap is closed (*obs2*).

$$
\begin{array}{ll}
\text{dom1} & Per(\mathsf{vol}(x)) \wedge Per(\mathsf{open}(x)) \wedge Per(\mathsf{cask}(x)) \wedge \\
 & Per(\mathsf{under}(x)) \wedge Dur(\mathsf{flow}(x,y),0) \\
\text{acs1} & [s,t]\mathsf{OpenTap}(x) \Rightarrow \\
 & \quad ([s]\mathsf{cask}(x) \Rightarrow R((s,t]\mathsf{open}(x))) \\
\text{acs2} & [s,t]\mathsf{CloseTap}(x) \Rightarrow \\
 & \quad ([s]\mathsf{cask}(x) \Rightarrow R((s,t]\neg\mathsf{open}(x))) \\
\text{dep1} & ([t]\mathsf{vol}(x)\hat{=}v \wedge \\
 & \quad [t{+}1]\mathsf{cask}(x) \wedge \mathsf{open}(x) \wedge \mathsf{under}(x)\hat{=}y) \Rightarrow \\
 & \quad I([t{+}1]\mathsf{flow}(x,y)\hat{=}(A_2\sqrt{2g\tfrac{v}{A_1}})) \\
\text{dep2} & [t]\mathsf{vol}(x)\hat{=}m \wedge sum(t{+}1,\mathsf{Flow}^{+}(t{+}1,x)) = n \wedge \\
 & \quad sum(t{+}1,\mathsf{Flow}^{-}(t{+}1,x)) = k \Rightarrow \\
 & \quad R([t+1]\mathsf{vol}(x)\hat{=}(m{+}n{-}k)) \\
\text{obs1} & [0]\mathsf{under}(\mathsf{cask1})\hat{=}\mathsf{jug1} \wedge \mathsf{cask}(\mathsf{cask1}) \\
\text{obs2} & [0]\forall x[\neg\mathsf{open}(x)] \\
\text{obs3} & [0]\mathsf{vol}(\mathsf{cask})\hat{=}100 \wedge \mathsf{vol}(\mathsf{jug})\hat{=}0 \\
\text{occ1} & [5,6]\mathsf{OpenTap}(\mathsf{cask1}) \\
\text{occ2} & [15,16]\mathsf{CloseTap}(\mathsf{cask1})
\end{array}
\tag{11}
$$

Opening the tap of the cask (*occ1*) produces a flow (flow(cask1,jug1)) from the cask to the jug that decreases as the level of beer in the cask decreases, and closing the tap (*occ2*) stops the flow. Notice that our representation of concurrency makes it easy to for instance open an additional cask above the jug; the total flow into the jug would then be the sum of flows from the casks.

4 Conclusions

We have presented a method for representing delayed effects of actions which utilizes previous results on ramification and concurrency. As a matter of fact, the formalism used here (TAL-C) is identical to the one presented in [64], but whereas that paper concentrated on the interaction between more or less immediate effects of actions, we here exploit the potential to deal with the additional dimension of delay. There are essentially three features of TAL-C that provide this potential. First, there is a notion of explicit time that is independent of action, and that allows us to easily formulate

temporal expressions such as "after 15 seconds". Second, there are dependency laws that allow us to describe delayed effects outside of action laws, in addition to dealing with complex ramifications. Third, there is a support for concurrency, which is based on the aforementioned dependency laws and a distinction between persistent and durational features. This permits the delayed effects of an action to interact with other actions. In fact, it is the complications that concurrency adds that is the tricky issue in representing delayed effects, and maybe this explains why so little progress has been made on the subject. Our examples have illustrated delayed effects with duration, interactions with other actions including interruptions, and modeling of actual processes. We should also emphasize the fact that the resulting theories are first-order. The examples in this paper have been implemented using a tool called VITAL available online at `http://anton.ida.liu.se/`. Future work obviously includes further exploration of delayed effects, concurrency and processes. In that context, work in qualitative reasoning [42, 76] will be of interest (in particular, the concept of an influence has been important).

Acknowledgments

This work has been supported by the Wallenberg foundation and by the Swedish Research Council for Engineering Sciences (TFR).

Paper III

Causal relations between actions in TAL

Lars Karlsson
Unpublished manuscript

Abstract

This paper presents an extension to TAL which supports reasoning about causal relations between action and event occurrences. It is shown how to represent actions that can be triggered by conditions in the world or by other actions, and actions that can be prevented by other actions. In addition, the time between the triggering action or condition and the triggered action can be indeterminate or only partially determined. Technically the extension involves introducing a new predicate for action occurrences to be used in trigger conditions in laws describing causal relations between actions. The minimization policy (global minimization of occurrences) is only slightly modified and retains the first-order reducibility of the original minimization policy of TAL.

1 Introduction

The work by Pinto [113], which was described in chapter 5, addressed the interesting topic of triggered actions and events. To be able to express causal relations between actions and events is a significant extension of the

ontology and to some extent also the epistemology (all occurrences need not be explicitly stated) for any logic of action and change. In this paper, we consider how to incorporate triggered actions and events in TAL. The fact that TAL has a single, metric time-line greatly simplifies this enterprise. Our investigation starts with the simple cases, and then we gradually introduce more complications.

As Pinto remarks [113], representations of actions that trigger actions can be seen as convenient abstractions of (sometimes complicated) processes. If our actions represent changes due some natural processes, then one can imagine alternative representations that do not make use of actions at all, but completely rely on for instance dependency laws.[1] On the other hand, there are also cases where the identity of the action that caused the change is significant. Consider for instance "if you shoot somebody, you will go to jail." In this case, it does not suffice to look at the effects of the shooting (a person dies and the gun becomes unloaded) to determine that an action (go to jail) is triggered. It should also be observed that in this example, the causal relation between shooting and going to jail is due to the attitudes and actions of other agents as they are expressed in for instance legislation and social organization.

There are a number of problems with triggered actions that are not addressed in this paper, and they are due to the fact that triggered actions might occur concurrently with other actions. As long as there are only atomic statements about action occurrences, it is straightforward to write scenarios where actions do not occur simultaneously. However, with triggered actions, this becomes inherently difficult. Concurrency is treated in paper I. A strongly related topic is that of delayed effects of actions. Often, a triggered event can be represented as a delayed effect and vice versa. Delayed effects are addressed in paper II.

The remainder of this paper presupposes that the reader has read the presentation of TAL in chapter 3. It is also advantageous to have read chapter 5, and in particular section 5.3.

[1] In SC, the concept of time is based on actions, so there cannot be any change without actions. This seems to be the main reason why Reiter [120] (among others) represents discontinuous changes of physical parameters (e.g. when a ball bounces) as natural actions. This is not necessary for logics with an independent notion of time (e.g. Sandewall's continuous-time logic [123]; see also paper II).

2 Actions triggered by conditions on fluents

First, we consider the simple case where an action is triggered by some condition becoming true in the world, instead of by other actions. One example is a slightly modified version of Pinto's Eden example [113]: "if the forbidden fruit is consumed, you will be (immediately) expelled." Taking some liberties regarding the form of an occurrence statement in TAL, this example can be encoded in the surface language $\mathcal{L}(ND)$ of TAL as follows.

$$occ1 \quad C_T([t]\mathsf{fruit\text{-}eaten}) \Rightarrow [t+\Delta, t+\Delta+5]\mathsf{Expel} \tag{1}$$

Notice the use of the C_T ("changes to true") operator in the antecedent. The action Expel is triggered by the feature fruit-eaten becoming true, and not once for every time-point the feature is true.

The translation of $occ1$ to $\mathcal{L}(FL)$ is as follows.

$$\begin{aligned} occ1 \quad & \forall t[\,(Holds(t, \mathsf{fruit\text{-}eaten}, \mathsf{T}) \wedge \\ & \quad \forall t'[t'+1 = t \Rightarrow \neg Holds(t', \mathsf{fruit\text{-}eaten}, \mathsf{T})]) \Rightarrow \\ & \quad Occurs(t+\Delta, t+\Delta+5, \mathsf{Expel})\,] \end{aligned} \tag{2}$$

In addition, assume that there is an action occurrence EatFruit.

$$occ2 \quad [5,6]\mathsf{EatFruit} \tag{3}$$

The translation of $occ2$ to $\mathcal{L}(FL)$ is as follows.

$$occ2 \quad Occurs(5, 6, \mathsf{EatFruit}) \tag{4}$$

In chapter 3, we mentioned that occurrence statements are minimized. This is due to the assumption that there is complete and correct information about action occurrences. Formally, the minimization policy used in TAL for action occurrences is as follows:

$$Circ_{SO}(\Gamma_{occ}(Occurs); Occurs).$$

This minimization policy is based on second-order circumscription, and implies that the predicate $Occurs$ is minimized relative to the occurrence statements of the narrative. The second-order circumscription of a number of predicates $\overline{P} = P_1, \ldots, P_n$ in the theory $\Gamma(\overline{P})$ is denoted $Circ_{SO}(\Gamma(\overline{P}); \overline{P})$ (see Lifschitz [82]). Intuitively, $Circ_{SO}(\Gamma(\overline{P}); \overline{P})$ represents a (second-order) theory containing $\Gamma(\overline{P})$ where the extensions of the predicates $\overline{P}$ are minimal.

The following theorem due to Lifschitz is the basis for a convenient equivalence between circumscription of $Occurs$ (and $Occlude$) and predicate completion.

Theorem 1 *[84] If $F(\overline{x})$ does not contain P, then*

$$Circ(\forall\overline{x}[F(\overline{x}) \Rightarrow P(\overline{x})]; P;) \equiv \forall\overline{x}[F(\overline{x}) \equiv P(\overline{x})].$$

We apply this minimization policy to $\{occ1, occ2\}$:

$$\begin{aligned} Circ_{SO}(\{ \forall t[(&Holds(t, \mathsf{fruit\text{-}eaten}, \mathsf{T}) \wedge \\ &\forall t'[t' + 1 = t \Rightarrow \neg Holds(t', \mathsf{fruit\text{-}eaten}, \mathsf{T})]) \Rightarrow \\ &Occurs(t + \Delta, t + \Delta + 5, \mathsf{Expel})], \\ &Occurs(5, 6, \mathsf{EatFruit})\}; Occurs;) \end{aligned} \tag{5}$$

We can rewrite (5) in an equivalent form compatible with the left-hand-side of the equivalence of theorem 1, as follows,

$$\begin{aligned} Circ_{SO}(\{ \forall t, t', a[(&(t' = t + 5 \wedge a = \mathsf{Expel} \wedge \\ &\exists s[t = s + \Delta \wedge Holds(s, \mathsf{fruit\text{-}eaten}, \mathsf{T}) \wedge \\ &\forall s'[s' + 1 = s \Rightarrow \neg Holds(s', \mathsf{fruit\text{-}eaten}, \mathsf{T})]]) \vee \\ &\langle t, t', a\rangle = \langle 5, 6, \mathsf{EatFruit}\rangle) \Rightarrow \\ &Occurs(t, t', a)]\}; Occurs;) \end{aligned} \tag{6}$$

Finally, theorem 1 can be used to establish that (6) is equivalent to the following.

$$\begin{aligned} &\forall t, t', a[((t' = t + 5 \wedge a = \mathsf{Expel} \wedge \\ &\exists s[t = s + \Delta \wedge Holds(s, \mathsf{fruit\text{-}eaten}, \mathsf{T}) \wedge \\ &\forall s'[s' + 1 = s \Rightarrow \neg Holds(s', \mathsf{fruit\text{-}eaten}, \mathsf{T})]]) \vee \\ &\langle t, t', a\rangle = \langle 5, 6, \mathsf{EatFruit}\rangle) \equiv \\ &Occurs(t, t', a)] \end{aligned} \tag{7}$$

Nothing has yet been stated about the effects of actions. For simplicity, just assume that EatFruit has the effect of making fruit-eaten true, and that no other action has that effect. In addition, assume that fruit-eaten is false at time-point 0. From (7), one can infer that $Occurs(5, 6, \mathsf{EatFruit})$ is true. Consequently, $Holds(6, \mathsf{fruit\text{-}eaten}, \mathsf{T})$ is true, while $Holds(t, \mathsf{fruit\text{-}eaten}, \mathsf{T})$ is false for $t < 6$ due to the persistence assumption. This in turn implies that the condition associated with Expel in (7) is true, and that $Occurs(6 + \Delta, 11 + \Delta, \mathsf{Expel})$ is true.

The procedure described above for obtaining the necessary and sufficient conditions for when an action occurs applies not just to the Eden example, but to any occurrence statement of the forms:

$$\begin{aligned} &occ \quad \alpha \Rightarrow [\tau, \tau']\mathsf{A} \\ &occ \quad [\tau, \tau']\mathsf{A} \end{aligned} \tag{8}$$

where α does not contain any action occurrence statements. In conclusion, actions triggered by conditions expressed in terms of fluents can be represented in TAL without any problems.

3 Positive determinate correlation, simple temporal relation

Next, we consider the simple case where there is a positive correlation between two types of occurrences and the temporal relation between them is determinate. One instance of this case is the Eden scenario: "if you eat the forbidden fruit, you will be expelled." It can be straight-forwardly encoded in TAL (in $\mathcal{L}(ND)$) as follows, where Δ is an integer.

$$occ1 \quad [s,t]\mathsf{EatFruit} \Rightarrow [t+\Delta, t+\Delta+5]\mathsf{Expel} \tag{9}$$

The translation of $occ1$ to $\mathcal{L}(FL)$ is as follows.

$$occ1 \quad Occurs(s,t,\mathsf{EatFruit}) \Rightarrow Occurs(t+\Delta, t+\Delta+5, \mathsf{Expel}) \tag{10}$$

In addition, assume that there is an action occurrence EatFruit.

$$occ2 \quad [5,7]\mathsf{EatFruit} \tag{11}$$

The translation of $occ2$ to $\mathcal{L}(FL)$ is as follows.

$$occ2 \quad Occurs(5,7,\mathsf{EatFruit}) \tag{12}$$

The circumscription policy presented in section 2 is also applicable to the Eden scenario, as follows.

$$Circ_{SO}(\ \{Occurs(s,t,\mathsf{EatFruit}) \Rightarrow Occurs(t+\Delta, t+\Delta+5, \mathsf{Expel}),\ Occurs(5,7,\mathsf{EatFruit})\}; Occurs;\ ) \tag{13}$$

A model M that is minimal with respect to the *Occurs* predicate can be constructed as follows (for simplicity, we assume that $\Delta = 1$.) First, if M is to be a model of (13), then $M \models Occurs(5,7,\mathsf{EatFruit})$. Second, it must then also be the case that $M \models Occurs(8,13,\mathsf{Expel})$. Thus, we have that any model M must satisfy

$$M \models \forall t,t',a[\ (\langle t,t',a\rangle = \langle 5,7,\mathsf{EatFruit}\rangle \vee \langle t,t',a\rangle = \langle 8,13,\mathsf{Expel}\rangle) \Rightarrow Occurs(t,t',a)]. \tag{14}$$

Now consider an interpretation M' such that

$$M' \models \forall t,t',a[\ (\langle t,t',a\rangle = \langle 5,7,\mathsf{EatFruit}\rangle \vee \langle t,t',a\rangle = \langle 8,13,\mathsf{Expel}\rangle) \equiv Occurs(t,t',a)]. \tag{15}$$

It can be verified that M' is a model. It should also be clear that M' is minimal with respect to *Occurs*. Only the two tuples $\langle 5, 7, \mathsf{EatFruit}\rangle$ and $\langle 8, 13, \mathsf{Expel}\rangle$, which we had established must belong to *Occurs*, do indeed belong to *Occurs*. That is to say, on a semantic level the intended conclusions are obtained.

Regarding the syntactic aspects, recall theorem 1. The Eden scenario (13) does not satisfy the condition on the left-hand-side. Consequently, there is no transformation of (13) to a form in which theorem 1 is applicable. We will soon return to this problem, and actually provide a solution.

4 Negative correlations

The first complication of the simple case in section 3 we consider is when the correlation between two types of occurrences is negative instead of positive. Consider this example: if Tom plays with his toys and doesn't put them back afterwards, his mother will be angry with him. Thus, the anger of Tom's mother is dependent on the absence of the occurrence of Tom putting back his toys. The following is an attempt to formalize this scenario, using the same methods as in the Eden scenario.

$$\begin{array}{ll} occ1 & [20, 80]\mathsf{Play} \\ occ2 & [s, s']\mathsf{Play} \wedge \neg\exists t, t'[s' \leq t \leq t' \leq 60 \wedge [t, t']\mathsf{PutBack}] \Rightarrow \\ & [s' + 60, s' + 65]\mathsf{Angry} \end{array} \tag{16}$$

The $\mathcal{L}(FL)$ version is as follows.

$$\begin{array}{ll} occ1 & \mathit{Occurs}(20, 80, \mathsf{Play}) \\ occ2 & (\mathit{Occurs}(s, s', \mathsf{Play}) \wedge \\ & \neg\exists t, t'[s' \leq t \leq t' \leq s' + 60 \wedge \mathit{Occurs}(t, t', \mathsf{PutBack})]) \Rightarrow \\ & \mathit{Occurs}(s' + 60, s' + 65, \mathsf{Angry}) \end{array} \tag{17}$$

Note that the antecedent of *occ*2 contains an *Occurs* in a negated subformula. As observed by Thielscher [137], this might yield unintended models. Why this is the case can be understood by looking at the following statement, which is equivalent to *occ*2.

$$\begin{array}{ll} occ2' & \mathit{Occurs}(s, s', \mathsf{Play}) \Rightarrow \\ & (\exists t, t'[s' \leq t \leq t' \leq s' + 60 \wedge \mathit{Occurs}(t, t', \mathsf{PutBack})] \vee \\ & \mathit{Occurs}(s' + 60, s' + 65, \mathsf{Angry})) \end{array} \tag{18}$$

If one minimizes *Occurs* in this theory, the result is that either Angry occurs and PutBack does not (which is intended), or PutBack occurs and Angry

does not (which is not intended). Thielscher [137] solves the occurrence minimization problem in a version of the fluent calculus by turning action occurrences into fluents and then relying on the causal machinery for fluents. For instance, the event occurrence $Happens(140, Angry)$ becomes the fluent statement $[140]happens(Angry)$. The fluent $happens(Angry)$ is a momentary fluent, which means that it normally is false, and it is only temporarily true under the immediate influence of some causal law. Turning actions into fluents is possible as action occurrences are instantaneous in Thielscher's approach. However, Thielscher's solution cannot be transfered directly to TAL, where action occurrences have extension. Instead, we shall introduce a new predicate $Occurs_o : \mathcal{T} \times \mathcal{T} \times \mathcal{A}$ that is used in the antecedents of conditional occurrence statements. The subscript o denotes here that the occurrence has already been observed or inferred from some other occurrence statement, as opposed to being caused or enforced by the current statement. The second part of our solution is to minimize $Occurs$ while keeping $Occurs_o$ fixed. Thereby, we assure that occurrence causality is applied in the intended direction. After minimization, we define $Occurs_o$ as being equivalent to $Occurs$ and filter the $Occurs$-minimized subtheory. The details of the approach are presented below, illustrated with the toy scenario.

The function $Trans$, which translates $\mathcal{L}(ND)$ statements to $\mathcal{L}(FL)$ statements (see paper I, appendix A), is extended as follows.

$$Trans([\tau, \tau']A_o) = Occurs_o(\tau, \tau', A) \tag{19}$$

The toy scenario is modified by introducing the o subscript in the antecedent of $occ2$. Thereby, we encode a causal direction from the playing and cleaning to the anger of the mother.

$$\begin{array}{ll} occ1 & [20, 80]\mathsf{Play} \\ occ2 & ([s, s']\mathsf{Play}_o \wedge \\ & \quad \neg\exists t, t'[s' \leq t \leq t' \leq s' + 60 \wedge [t, t']\mathsf{PutBack}_o]) \Rightarrow \\ & \quad [s' + 60, s' + 65]\mathsf{Angry} \end{array} \tag{20}$$

The $\mathcal{L}(FL)$ versions of $occ1$ and $occ2$ are as follows.

$$\begin{array}{ll} occ1 & Occurs(20, 80, \mathsf{Play}) \\ occ2 & (Occurs_o(s, s', \mathsf{Play}) \wedge \\ & \quad \neg\exists t, t'[s' \leq t \leq t' \leq s' + 60 \wedge \\ & \quad\quad Occurs_o(t, t', \mathsf{PutBack})]) \Rightarrow \\ & \quad Occurs(s' + 60, s' + 65, \mathsf{Angry}) \end{array} \tag{21}$$

Observe that (21) can be rewritten in a way compatible with theorem 1:

$$\begin{array}{l}\forall t,t',a[(\,\langle t,t',a\rangle = \langle 20,80,\mathsf{Play}\rangle \vee \\ \ (a=\mathsf{Angry} \wedge \\ \exists s,s'[Occurs_o(s,s',\mathsf{Play}) \wedge t+s'=60 \wedge t'=s'+65 \wedge \\ \ \neg\exists u,u'[s'\le u\le u'\le s'+60 \wedge Occurs_o(u,u',\mathsf{PutBack})]\,]\,)\,) \Rightarrow \\ \ Occurs(t,t',a)\,].\end{array} \tag{22}$$

Consequently, minimizing $Occurs$ while keeping $Occurs_o$ fixed yields the following result.

$$\begin{array}{l}Circ_{SO}(\Gamma_{occ};Occurs;) \\ = \\ \Gamma_{occ} \wedge \\ \forall t,t',a[Occurs(t,t',a) \equiv \\ \quad (\,\langle t,t',a\rangle = \langle 20,80,\mathsf{Play}\rangle \vee \\ \quad\ (a=\mathsf{Angry} \wedge \exists s,s'[Occurs_o(s,s',\mathsf{Play}) \wedge t+s'=60 \wedge t'=s'+65 \\ \quad\ \wedge\neg\exists u,u'[s'\le u\le u'\le s'+60 \wedge Occurs_o(u,u',\mathsf{PutBack})]\,]\,)\,)\end{array} \tag{23}$$

Finally, (23) should be filtered with the following axiom in order to relate $Occurs$ to $Occurs_o$:

$$\forall t,t',a[Occurs(t,t',a) \equiv Occurs_o(t,t',a)]. \tag{24}$$

Axiom (24) states that the only actions that occur are those that are observed or indirectly inferred. Given (23) and (24), one can now deduce

$$Occurs(20,80,\mathsf{Play}) \wedge Occurs(140,145,\mathsf{Angry}) \tag{25}$$

as follows. First, $Occurs(20,80,\mathsf{Play})$ follows from (23). From this and (24), one can infer $Occurs_o(20,80,\mathsf{Play})$. Further, from (23) one can infer

$$\neg\exists u,u'[80\le u\le u'\le 140 \wedge Occurs(u,u',\mathsf{PutBack})] \tag{26}$$

which together with (24) yields

$$\neg\exists u,u'[80\le u\le u'\le 140 \wedge Occurs_o(u,u',\mathsf{PutBack})]. \tag{27}$$

From the latter statement and $Occurs_o(20,80,\mathsf{Play})$, one can finally infer $Occurs(140,145,\mathsf{Angry})$. Consequently, we have shown that

$$Occurs(20,80,\mathsf{Play}) \wedge Occurs(140,145,\mathsf{Angry}) \tag{28}$$

is a consequence of the narrative.

In summary, the technique for representing triggered action occurrences whose trigger conditions might involve negative occurrences is as follows.

1. First, the triggered action occurrences must be made distinct from the triggers, in order to assure that causality works in the right direction. This is achieved by introducing the $Occurs_o$ predicate, which is used for triggers.

2. The minimization policy is $Circ_{SO}(\Gamma_{occ}; Occurs;)$. It minimizes $Occurs$ (actions that are caused) while keeping $Occurs_o$ fixed (actions that are causes). As $Occurs$ only appears on the right-hand-side of implications, it is ensured that there is a first-order reduction of the circumscribed theory. It even implies, due to theorem 1, that the minimization policy is equivalent to predicate completion.

3. Finally, the two predicates $Occurs_o$ and $Occurs$ need to be connected. This is done by filtering with the sentence

$$\Gamma_{oo} = \forall t, t', a[Occurs(t, t', a) \equiv Occurs_o(t, t', a)].$$

Consequently, the circumscription policy with filtering is:

$$Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo} \tag{29}$$

5 Positive correlations revisited

An important element of the technique described in the previous section is that the condition expressed in theorem 1 is satisfied. This is why the circumscription in (23) resulted in a convenient reduction. Note that this technique would also solve the problems encountered in connection with the Eden scenario in the previous section. It just has to be modified as follows (we assume that $\Delta = 1$),

$$\begin{array}{ll} occ1 & [s,t]\mathsf{EatFruit}_o \Rightarrow [t+1, t+6]\mathsf{Expel} \\ occ2 & [5,7]\mathsf{EatFruit}. \end{array} \tag{30}$$

We skip the details, and simply conclude that circumscription of $Occurs$ with filtering yields the following result.

$$\begin{array}{l} Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo} \; = \\ \Gamma_{occ} \wedge \\ \forall t, t', a[Occurs(t, t', a) \equiv \\ \quad (\langle t, t', a\rangle = \langle 5, 7, \mathsf{EatFruit}\rangle \vee \\ \quad\; (a = \mathsf{Expel} \wedge t' = t + 5 \wedge \\ \quad\quad \exists s, s'[s' + 1 = t \wedge Occurs_o(s, s', \mathsf{EatFruit})]))] \wedge \\ \forall t, t', a[Occurs(t, t', a) \equiv Occurs_o(t, t', a)]. \end{array} \tag{31}$$

From (31), one can conclude that $Occurs(5, 7, \mathsf{EatFruit})$ is true, and therefore also $Occurs_o(5, 7, \mathsf{EatFruit})$. From this, it follows that $Occurs(8, 13, \mathsf{Expel})$, as intended.

In summary, the use of $Occurs_o$ both provides a convenient syntactic form when one circumscribes composite occurrence statements, and it solves semantic problems due to negative occurrences in triggers.

The method has one restriction, though. Assume that there are two actions that can instantaneously trigger each other (or that have to be performed together), as in the following scenario.

$$\begin{array}{ll} occ1 & [s,t]\mathsf{Eat}_o \Rightarrow [s,t]\mathsf{Drink} \\ occ2 & [s,t]\mathsf{Drink}_o \Rightarrow [s,t]\mathsf{Eat} \end{array} \tag{32}$$

In this case, the circumscribed theory has the following form (omitting intermediary steps):

$$\begin{array}{l} Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo} \ = \\ \Gamma_{occ} \wedge \\ \forall t, t', a[Occurs(t, t', a) \equiv \\ \quad (a = \mathsf{Eat} \wedge Occurs_o(t, t', \mathsf{Drink}) \vee \\ \quad\ a = \mathsf{Drink} \wedge Occurs_o(t, t', \mathsf{Eat}))] \wedge \\ \forall t, t', a[Occurs(t, t', a) \equiv Occurs_o(t, t', a)]. \end{array} \tag{33}$$

The conclusion is

$$\forall t, t'[Occurs(t, t', \mathsf{Drink}) \equiv Occurs(t, t', \mathsf{Eat})]$$

but nothing can be said about whether the two actions actually occur or not. For each pair of time-points, there are models where both actions do not occur, and models where they do. The latter corresponds to a spontaneous mutual triggering of the two actions. This is clearly not a desired feature. Consequently, cyclic instantaneous causal relations between actions as in (32) should be prohibited.

6 Partially determined and indeterminate time

So far, the temporal relations between actions have been determinate. Now, we consider cases where the time it takes for a triggered action or event to occur is indeterminate. For instance, assume that the Eden example is

modified to allow partially determined durations between the eating of the fruit and the expulsion.

$$\begin{array}{ll} occ1 & [s,t]\mathsf{EatFruit}_o \Rightarrow \\ & \quad \exists s',t'[\,[s',t']\mathsf{Expel} \wedge t+5 \leq s' \leq t+10 \wedge t' \leq s'+10\,] \\ occ2 & [5,7]\mathsf{EatFruit} \end{array} \tag{34}$$

The translation to $\mathcal{L}(SD)$ is as follows.

$$\begin{array}{ll} occ1 & \forall s,t[Occurs_o(s,t,\mathsf{EatFruit}) \Rightarrow \\ & \quad \exists s',t'[Occurs(s',t',\mathsf{Expel}) \wedge \\ & \qquad t+5 \leq s' \leq t+10 \wedge t' \leq s'+10]\,] \\ occ2 & Occurs(5,7,\mathsf{EatFruit}) \end{array} \tag{35}$$

Now, let us consider what happens when the circumscription policy

$$Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo}$$

is applied to this scenario. Semantically, it can be verified that the *Occurs*-minimal models are those where each EatFruit is followed by exactly one Expel between 5 and 10 time steps later. However, syntactically, the existential on the right-hand-side in $occ1$ causes some problems. The scenario is not in a form compatible with theorem 1, so the circumscriptive policy cannot be reduced to the use of predicate completion.

A potential solution could be to eliminate the existential by using two constants $\mathbf{n}_1$ and $\mathbf{n}_2$ for representing the variable delay, as follows.

$$\begin{array}{ll} occ1 & [s,t]\mathsf{EatFruit}_o \Rightarrow [t+\mathbf{n}_1, t+\mathbf{n}_2]\mathsf{Expel} \\ obs1 & 5 \leq \mathbf{n}_1 \leq 10 \wedge \mathbf{n}_1 < \mathbf{n}_2 \leq \mathbf{n}_1 + 10 \\ occ2 & [5,7]\mathsf{EatFruit} \end{array} \tag{36}$$

Circumscribing *Occurs* in the $\mathcal{L}(SD)$ translation of (36) yields the following result.

$$\begin{array}{l} Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo} \;= \\ \Gamma_{occ} \wedge \\ \forall t,t',a[\; Occurs(t,t',a) \equiv \\ \qquad (\,\langle t,t',a\rangle = \langle 5,7,\mathsf{EatFruit}\rangle \vee \\ \qquad\; (a = Expel \wedge t' = t+5 \wedge \\ \qquad\quad \exists s,s'[Occurs_o(s,s',\mathsf{EatFruit}) \wedge \\ \qquad\qquad s'+\mathbf{n}_1 = t \wedge s'+\mathbf{n}_2 = t']\,)\,] \wedge \\ \qquad \forall t,t',a[Occurs(t,t',a) \equiv Occurs_o(t,t',a)]. \end{array} \tag{37}$$

There is a problem with this solution, though. It is true that $\mathbf{n}_1$ and $\mathbf{n}_2$ might have different values in different models, but within a specific model,

they can each have only one specific value. Consequently, if there are several occurrences of fruit eating over time, each of these will be followed by an expulsion with exactly the same delay. The delay is only partially specified, but never changes.[2]

Studying the version of the Eden scenario with constant delays again (36), one observes that there is an implicit quantification over s and t, and indirectly also over occurrences of EatFruit, which does not involve the delay constants $\mathbf{n}_1$ and $\mathbf{n}_2$. This observation suggests representing the delay as functions of the action occurrence instead of as constants. The functions $\mathbf{n}(s,t,\mathsf{EatFruit})$ and $\mathbf{n}'(s,t,\mathsf{EatFruit})$ represent the start and end times of the action triggered by $[s,t]\mathsf{EatFruit}$. The following is (36) modified to use delay functions.

$$
\begin{array}{ll}
occ1 & [s,t]\mathsf{EatFruit}_o \Rightarrow \\
 & \quad ([\mathbf{n}(s,t,\mathsf{EatFruit}), \mathbf{n}'(s,t,\mathsf{EatFruit})]\mathsf{Expel} \wedge \\
 & \quad\ s+5 \leq \mathbf{n}(s,t,\mathsf{EatFruit}) \leq s+10 \wedge \\
 & \quad\ \leq \mathbf{n}(s,t,\mathsf{EatFruit}) < \mathbf{n}'(s,t,\mathsf{EatFruit}) \leq \mathbf{n}(s,t,\mathsf{EatFruit}) + 10) \\
occ2 & [5,7]\mathsf{EatFruit}
\end{array}
\tag{38}
$$

Circumscribing *Occurs* yields the following result.

$$
\begin{array}{l}
Circ_{SO}(\Gamma_{occ}; Occurs;) \wedge \Gamma_{oo} \ = \\
\Gamma_{occ} \wedge \\
\forall t,t',a[\ Occurs(t,t',a) \equiv \\
\qquad (\langle t,t',a\rangle = \langle 5,7,\mathsf{EatFruit}\rangle \vee \\
\qquad\ (a = Expel \wedge \exists s,s'[Occurs_o(s,s',\mathsf{EatFruit}) \wedge \\
\qquad\quad t = \mathbf{n}(s,s',\mathsf{EatFruit}) \wedge t' = \mathbf{n}'(s,s',\mathsf{EatFruit})\,])\,)\,] \wedge \\
\qquad \forall t,t',a[Occurs(t,t',a) \equiv Occurs_o(t,t',a)].
\end{array}
\tag{39}
$$

It should be pointed out, though, that semantically, the use of existential quantifiers (34) and delay functions (38) do not produce identical results. For instance, assume that there are two adjacent occurrences of EatFruit

$$
\begin{array}{ll}
occ2 & [5,7]\mathsf{EatFruit} \\
occ3 & [8,9]\mathsf{EatFruit}
\end{array}
\tag{40}
$$

[2] Actually, in this particular scenario, it might be reasonable to assume that there is only one fruit-eating action. This assumption does not hold in general, though.

Let M be a model where the following hold,

$M \models Occurs(5, 7, \mathsf{EatFruit})$
$M \models Occurs(8, 9, \mathsf{EatFruit})$
$M \models \neg Occurs(t, t', \mathsf{EatFruit})$ for all remaining time-points $t,\ t'$.
$M \models Occurs(14, 16, \mathsf{Expel})$
$M \models Occurs(17, 20, \mathsf{Expel})$
$M \models \neg Occurs(t, t', \mathsf{Expel})$ for all remaining time-points $t,\ t'$.
$M \models \mathbf{n}(5, 7, \mathsf{EatFruit}) = 14 \wedge \mathbf{n}'(5, 7, \mathsf{EatFruit}) = 16$
$M \models \mathbf{n}(8, 9, \mathsf{EatFruit}) = 17 \wedge \mathbf{n}'(8, 9, \mathsf{EatFruit}) = 20$

M is a minimal model of (38) with delay functions, extended with (40), but it is not a minimal model of (34). In the latter case, there is a model with $M \not\models Occurs(17, 20, \mathsf{Expel})$ which has a smaller extension of $Occurs$.

From a notational perspective, a less attractive feature of the solution with delay functions is the clumsiness of the resulting temporal terms. Although the use of complex and long terms cannot be entirely avoided, it would be advantageous not to have to write them out so many times. This can be achieved as follows. First, consider the following logically equivalent reformulation of $occ1$ in (38),

$$\begin{array}{l}[s, t]\mathsf{EatFruit}_o \Rightarrow \\ \exists s', t'[\ s' = \mathbf{n}(s, t, \mathsf{EatFruit}) \wedge t' = \mathbf{n}'(s, t, \mathsf{EatFruit}) \wedge \\ \quad [s', t']\mathsf{Expel} \wedge s + 5 \leq s' \leq s + 10 \wedge t' \leq s' + 10].\end{array} \tag{41}$$

Next, the notation can be made even more compact with the following abbreviation,

$$\exists_{\phi}\mathbf{s}, \mathbf{t}[\gamma] \ =_{def}\ \exists \mathbf{s}, \mathbf{t}[s' = \mathbf{n}\phi \wedge t' = \mathbf{n}'\phi \wedge \gamma]. \tag{42}$$

Now, (41) can be written in the following compact form,

$$\begin{array}{l}[s, t]\mathsf{EatFruit}_o \Rightarrow \\ \exists_{(s,t,EatFruit)} s', t'[\ [s', t']\mathsf{Expel} \wedge \\ \quad s + 5 \leq s' \leq s + 10 \wedge t' \leq s' + 10].\end{array} \tag{43}$$

As (43) is logically equivalent to (38)[3] and it is straight-forward to make the transformation from the former to the latter before circumscribing, Theorem 1 is still applicable.

[3]Observe that the equivalence holds in the general case, i.e. $\alpha(\tau, \tau') \equiv \exists s, s'[s = \tau \wedge s' = \tau' \wedge \alpha(s, s')]$, and not just for this particular scenario. From left to right: assume $\alpha(\tau, \tau')$, which gives $(\tau = \tau \wedge \tau' = \tau' \wedge \alpha(\tau, \tau'))$ (by conjunctive introduction) which gives $\exists s, s'[s = \tau \wedge s' = \tau' \wedge \alpha(s, s')]$ (existential introduction). From right to left: assume $\exists s, s'[s = \tau \wedge s' = \tau' \wedge \alpha(s, s')]$. Assume $[a = \tau \wedge b = \tau' \wedge \alpha(a, b)]$ for two previously unused constants a and b. As $a = \tau$ and $b = \tau'$, from $\alpha(a, b)$ follows $\alpha(\tau, \tau')$. As a and b do not appear in $\alpha(\tau, \tau')$, it holds also outside the last assumption (existential elimination).

7 Related work

In chapter 5, we have already described the approaches of Reiter [120] and Pinto [113] to triggered actions and events. In particular, Pinto addressed preventable, triggered and partially specified occurrences. In this paper, we mentioned Thielscher's [137] approach, which is to turn action occurrences into momentary fluents and take advantage of the causal machinery for fluents.

The motivated action theory (MAT) by Stein and Morgenstern [133] can also represent causal relations between actions. MAT is based on an explicit notion of when an action is motivated. A causal rule in MAT has the form

$$\alpha \wedge \beta \supset \gamma$$

where α is set of occurrence terms — the set of triggering events of the causal rule; β is a conjunction of precondition terms; and γ describes the result. Note that γ can include occurrence terms. There is a condition that all time-points in γ should be later than all time-points in $\alpha \wedge \beta$:

$$\exists t. \forall \xi \in \alpha \cup \beta, TIME(\xi) \leq t \text{ and } TIME(\gamma) \geq t$$

The following is an example of a causal rule,

$$\begin{aligned} &\forall t. OCCURS(t, \mathit{PressButton}) \wedge HOLDS(t, \mathit{credit}) \supset \\ &\quad HOLDS(t+1, \mathit{has\text{-}coffee}) \wedge HOLDS(t+1, \mathit{not}(\mathit{credit})). \end{aligned} \tag{44}$$

MAT is an explicitly syntactic theory, and the form of a causal rule is of significance in both the model theory and proof theory of the system. In particular, the model theory is based on an explicit notion of motivation: a statement λ is motivated in a model M if it is either explicitly stated or appears in the result component γ of a causal rule in which the action occurrences in α are motivated in M and β is true in M.[4] Stein and Morgenstern define a preference criteria on models in which models with fewer (in terms of subset) unmotivated occurrences are preferred. The definition of motivation is quite complex and involves meta-level conditions referring to the syntactic form of statements and the entailment relation.

What makes MAT interesting from the perspective of triggered actions is that the γ part of a causal rule can include occurrence terms. Stein and

[4] Stein and Morgenstern distinguish between four kinds of motivation: strong motivation in which λ holds in all models of a theory; weak motivation in which λ is motivated by a causal rule in the particular model M; semi-motivation in which a statement λ is a disjunct in the γ part of a causal rule; and existential motivation in which a statement λ is obtained by instantiating an existential in the γ part of a causal rule.

Morgenstern provide two examples of triggered actions. The first one is a line of dominos. The scenario contains a causal rule

$$\begin{array}{l}\forall t, 0 < i < n.(\, OCCURS(t, fall(domino_i)) \wedge \\ \quad \neg OCCURS(t, blockFall(domino_i))\,) \supset \\ \quad OCCURS(t+1, fall(domino_{i+1}))\end{array} \tag{45}$$

and a fact

$$OCCURS(1, fall(domino_1)). \tag{46}$$

The conclusion (due to the preference criteria — the preferred models only contain action occurrences motivated by (45)) is that all dominos fall,

$$\forall 0 < i \leq n. OCCURS(i, fall(domino_i)). \tag{47}$$

The second example used is a sunrise;

$$\forall t. HOLDS(t, daybreak) \supset OCCURS(t+1, sunrise). \tag{48}$$

Due to the scarcity of examples, it is difficult to determine exactly what are the possibilities and limitations of the approach. The form of causal rules and the examples above indicate that MAT can represent that actions are triggered by conditions in the world, or are triggered or prevented by other actions, or combinations thereof. In this respect, MAT is quite similar to TAL with triggered actions, except for that MAT permit disjunctions in the effect part; that is not possible for triggered actions in TAL. Also note that in MAT the condition that the effect should be strictly after the triggering events and preconditions excludes instantaneously triggered actions. TAL does not have this restriction. Actions in MAT have no extension, whereas they have extension in TAL.

It is interesting to observe the relation between $Occurs_o$ in TAL with triggered actions and the notion of a motivated action in MAT. Essentially, $Occurs_o$ represents actions that have been motivated, whereas *Occurs* is used to state that actions actually occur. By equating $Occurs_o$ with *Occurs* in TAL, one states that only motivated actions are assumed to occur; this is a stronger assumption than in MAT, where a minimal set of unmotivated actions are assumed to occur.

Finally, one important difference between TAL and MAT is the nature of the semantics and proof theory; TAL is based on first-order logic (with reservations for the temporal domain) whereas MAT has a complex, nonstandard semantics and proof theory based on different notions of motivation of facts.

8 Conclusions

In this paper, we have shown how to represent causal dependencies between actions in TAL in the sense that the occurrences or non-occurrences of certain actions can trigger the occurrences of other actions. This is accomplished by introducing a new predicate $Occurs_o$ to TAL, and the minimization policy is modified by adding a filter

$$\forall t, t', a[Occurs(t, t', a) \equiv Occurs_o(t, t', a)].$$

The extension is capable of representing the following:

- Actions can be triggered by conditions in the world.
- Actions can be triggered by other actions (positive correlation).
- The triggering of an action can be prevented by the occurrence of another action (negative correlation).
- The time between the triggering action or condition and the triggered action can be indeterminate or only partially determined.

Although the subject has been given some treatment by other authors such as Pinto [113] and Thielscher [137], what is presented here is novel in two important ways.

- Actions can have extension (a feature of original TAL).
- The temporal relations between actions can be partially determined or indeterminate.

The simplicity of the circumscription policy and its first-order reducibility should also be emphasized.

Paper IV

Reasoning with Incomplete Initial Information and Nondeterminism in Situation Calculus

Lars Karlsson
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
larka@ida.liu.se

Abstract

Situation Calculus is arguably the most widely studied and used formalism for reasoning about action and change. The main reason for its popularity is the ability to reason about different action sequences as explicit objects. In particular, planning can be formulated as an existence problem. This paper shows how these properties break down when incomplete information about the initial state and nondeterministic action effects are introduced, basically due to the fact that this incompleteness is not adequately manifested on the object level. A version of Situation Calculus is presented which adequately models the alternative ways the world can develop relative to a choice of actions.

1 Introduction

The contribution of this paper is to highlight some problems that arise in Situation Calculus when incomplete initial information or nondeterministic actions are considered, in particular when reasoning about plans. A new version of Situation Calculus is proposed that avoids these problems by containing incompleteness and nondeterminism within individual logical models just like different choices of actions are.

Situation Calculus [97, 118] is a formalism for reasoning about action and change that has been widely studied and applied to a broad range of problems. Its major strength relative to other formalisms is the possibility to perform a large extent of reasoning on the object level. Object-level reasoning occurs completely *within* the object language and within one single theory as opposed to meta-level reasoning which involves reasoning *about* theories. The advantage of object-level reasoning is that it is completely supported by the deductive system of the logic in use. Problem solving is equivalent to finding logical proofs.

There are variations between different formalisms for action and change in how much reasoning can be done on the object level. On one extreme, there are STRIPS-style formalisms [39], where even the state transition resulting from the application of an action is specified on the meta-level, namely as adding and deleting sentences to/from a state description. In the middle there are narrative-based formalisms such as Event Calculus [69] and Sandewall's logics [124], where each theory Γ represents a specific choice of actions. At the other end of the scale there are branching formalisms such as Situation Calculus. The use of a branching time structure implies that different courses of actions can be contained within the same theory.

In Situation Calculus, henceforth referred to as SC, the time-structure is built up by successive applications of actions, starting from the initial situation S_0. Each action application results in a new situation. Situations have names specifying via what sequence of actions they are reached, for instance $do(Load, S_0)$, $do(Fire, S_0)$ and $do(Fire, do(Load, S_0))$. The fact that a fluent f holds in a situation s is expressed as $Holds(f, s)$, for instance $\neg Holds(alive,\ do(Fire, do(Load, S_0)))$. The effects of actions are specified

using axioms such as the following.[1]

$$Poss(Fire, s) \wedge Holds(loaded,\ s) \Rightarrow \\ (\neg Holds(loaded,\ do(Fire, s)) \wedge \\ \neg Holds(alive,\ do(Fire, s))\ ). \tag{1}$$

The predicate $Poss(a, s)$ denotes the conditions under which a is possible to execute, for instance $Poss(Fire, s) \equiv Holds(has\_gun, s)$. The strength of this representation is that given a SC theory Γ with action effect axioms, all possible different action sequences and their effects are contained within Γ. Situations are first-order objects, and thus reasoning about different situations (i.e. different action sequences) can be done on the object level, deductively. In particular, a planning problem can be straightforwardly represented as an existence problem, whereas in narrative-based formalisms planning has to be formulated as an abductive, and thus metalogical, problem. Let Γ be an SC theory after the appropriate nonmonotonic preprocessing for minimizing change (for instance use of the explanation closure techniques in [118]), and let $G(s)$ be a goal. In addition, let $Exec(s)$ represent the fact that the situation s is reachable from the initial situation via a sequence of executable actions. Formally expressed, $Exec(s) \equiv [s = S_0 \vee \exists a, s'.s = do(a, s') \wedge Poss(a, s') \wedge Exec(s')]$. Then the problem of finding a plan that achieves the goal is equivalent to proving the following.

$$\Gamma \vdash \exists s.\, [G(s) \wedge Exec(s)] \tag{2}$$

The binding of the s that satisfies $G(s) \wedge Exec(s)$ is the plan. For instance, if one assumes a proper theory Γ for loading and shooting, a solution to the problem

$$\Gamma \wedge \neg Holds(loaded, S_0) \wedge Holds(alive, S_0) \vdash \\ \exists s.\, [\neg Holds(alive, s) \wedge Exec(s)] \tag{3}$$

is found in the binding $s = do(Fire, do(Load, S_0))$.

The fact that action sequences (situations) are objects in SC, which makes it possible to perform reasoning tasks like planning purely deductively, has been emphasized as the major advantage of SC relative to narrative-based formalisms (see for instance [120]). This has motivated a substantial amount of work attempting to incorporate reasoning about metric time and

[1] For notational convenience, any variable occurring free in a formula is assumed to be universally quantified. For instance, in the axiom for $Fire$ there is an implicit quantifier $\forall s$ prefixing the formula.

temporal relations into SC (see [120] again), which is one of the strong points of narrative-based formalisms. Unfortunately, the advantage of SC breaks down when the implicit assumptions of information completeness and determinism are abandoned. Section 2 demonstrates this problem. Section 3 presents a version of SC which deals correctly with incomplete information and nondeterminism on the level of individual actions, and section 4 introduces constructs for action composition. Section 5 presents some conclusions and comparisons.

2 Problems with incomplete information

There is a strong assumption regarding complete information of the initial situation in the traditional SC formulation of the planning problem. To see why this is the case, consider the following scenario where an agent has to choose between two doors. One of the doors leads to freedom ($free$), whereas one leads to death ($\neg alive$) in the shape of a vicious cobra. Let lf denote that the left door leads to freedom, and assume that $Holds(alive, S_0) \wedge \neg Holds(free, S_0)$ but lf is unspecified at S_0. There are two actions $GoLeft$ and $GoRight$ defined as follows:

$$\begin{array}{l} Poss(GoLeft, s) \equiv Holds(alive, s), \\ Poss(GoLeft, s) \Rightarrow \\ (Holds(lf, s) \Rightarrow Holds(free, do(GoLeft, s)) \wedge \\ \neg Holds(lf, s) \Rightarrow \neg Holds(alive, do(GoLeft, s))) \end{array} \tag{4}$$

$$\begin{array}{l} Poss(GoRight, s) \equiv Holds(alive, s), \\ Poss(GoRight, s) \Rightarrow \\ (\neg Holds(lf, s) \Rightarrow Holds(free, do(GoRight, s)) \\ \wedge \\ Holds(lf, s) \Rightarrow \neg Holds(alive, do(GoRight, s))) \end{array} \tag{5}$$

The following inference is correct (again, Γ is the theory after the appropriate nonmonotonic preprocessing)

$$\begin{array}{l} \Gamma \wedge Holds(alive, S_0) \wedge \neg Holds(free, S_0) \vdash \\ \quad \exists s.\, [Holds(free, s) \wedge Exec(s)] \end{array} \tag{6}$$

In the case where $Holds(lf, S_0)$, the binding $s = do(GoLeft, S_0)$ proves the existential, and in the case where $\neg Holds(lf, S_0)$, $s = do(GoRight, S_0)$ proves the existential (Fig. 1). However, neither of these can be considered a plan. The problem is that different initial situations are realized in different

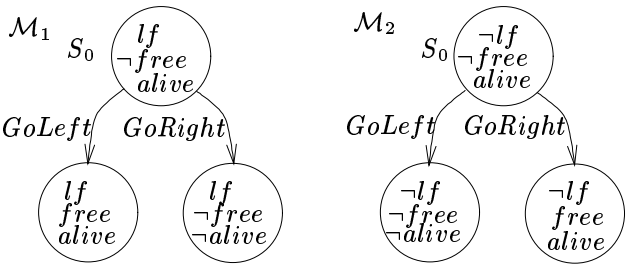

Figure 1: In model $\mathcal{M}_1$, $do(GoLeft, S_0)$ passes as a plan, and in $\mathcal{M}_2$, $do(GoRight, S_0)$ passes as a plan.

models, whereas the statement $\exists s.[Holds(free, s) \wedge Exec(s)]$ should be true in each individual model. Thus, it suffices to find one plan for each model; the plans can be different in different models. Statement (2) states that "for each possible initial situation one should find a plan that leads to the goal". The correct formulation should be "find a plan that leads to the goal for each possible initial situation". Unfortunately, the latter is not possible to express in SC. Even quantifying over initial situations like in

$$\forall s_0 .[Holds(alive, s_0) \wedge \neg Holds(free, s_0) \Rightarrow \exists s.[Holds(free, s) \wedge Exec(s_0, s)]\,], \tag{7}$$

just repeats the same erroneous formulation as above ($Exec(s_0, s)$ denotes that there is a sequence of executable actions from s_0 to s). The dual problem of negative plan existence gives rise to similar difficulties.

$$\neg\exists s.[Holds(free, s) \wedge Exec(s)]$$

is clearly an inadequate formulation of negative plan existence for the two-doors scenario. Note that it contradicts (6). Similar counter-examples can be constructed where incompleteness is due to an action with nondeterministic effects. As the action, due to the *do* function, just has one resulting situation in each model, it suffices to find one plan for each model.

In summary, when the implicit assumptions of complete initial information and determinism are abandoned, severe problems arise for SC. The reasons are the identification of plans with situations and, in the case of nondeterminism, the use of a *do* function. Thus, the critique presented here applies to all versions of SC that have any of these two features. As a consequence, it is not possible to express properties such as positive and negative plan existence on the object level, and this severely limits the usefulness of SC as a tool for analysis and specification of higher-level reasoning.

3 Actions and their effects

In this and the following sections, we will introduce a new version of SC. It is a state-based approach, where the frame problem is solved in a way similar to the PMON logic by Sandewall [124, 26]. PMON is a narrative-based logic with integer time and has been formally assessed correct for sequential worlds with actions with context-dependent and nondeterministic effects.

The following types are used: $\mathcal{S}$ for situations, $\mathcal{F}$ for fluents, $\mathcal{O}$ for objects in the domain and $\mathcal{A}$ for primitive actions. The two predicates $Holds(f, s)$ and $Occl(f, a, s)$ respectively denote that fluent f is true in situation s, and that if action a is executed in s, then f might be changed in the next state. *Occl* stands for "occluded" and denotes an explicit exception to the general principle of no-change. Notice that *Occl* represents a potential to change, not a necessity.

An essential feature of the logic is that the $do(a, s)$ function representing the result of doing a is replaced by a predicate $Res(s, a, s')$. This makes it possible to represent nondeterminism within individual models; if a behaves nondeterministically in situation s then there are more than one s' such that $Res(s, a, s')$. In previous approaches to nondeterminism in SC (e.g. [88]), $do(a, s)$ has denoted different situations in different models.

3.1 Elements of a theory

A theory consists of a set of axioms for unique names of objects, fluents and actions (UNA), and the following axioms.

Situation existence (SE). A second-order situation existence axiom (SE) defines the space of situations. In Baker's state-based approach [9], a situation existence axiom was used to obtain a correct minimization of change. In the approach presented here, there are no *do* terms denoting situations. Thus, without SE there might be models where some relevant situations are missing or even where the situation space is completely empty (an example of the former is given later in Fig. 2). SE states that for each possible state, that is to say each truth assignment Φ over the set of fluents, there is a corresponding situation s. In addition, all situations are unique.

$$\forall \Phi. \exists s. \forall f. \Phi(f) \equiv Holds(f, s) \tag{8}$$

$$\forall s, s'. \forall f. (Holds(f, s) \equiv Holds(f, s')) \Rightarrow s = s' \tag{9}$$

If the set of fluents is finite, a first-order SE is possible [9].

Action effects (LAW). To get a compact representation of action effects, a *Cause* macro is used. *Cause* involves both a statement that a fluent is allowed to change, and a statement of the result of this change.

$$Cause(s,a,s',f,T) =_{def} Occl(f,a,s) \wedge (Res(s,a,s') \Rightarrow Holds(f,s')) \tag{10}$$

$$Cause(s,a,s',f,F) =_{def} Occl(f,a,s) \wedge (Res(s,a,s') \Rightarrow \neg Holds(f,s')) \tag{11}$$

The effects of actions are specified in a set LAW. The normal form of an effect law for an action a is as follows.

$$\forall s,s'.\ \bigwedge_i (\ \alpha_i(s) \Rightarrow \bigvee_j (\bigwedge_k Cause(s,a,s',f_{ik},V_{ijk}))) \tag{12}$$

The formulae $\alpha_i(s)$ should only contain *Holds* expressions relating to s, and $V_{ijk} \in \{F,T\}$. The consequents are disjunctions of conjunctions, where each conjunction represents one possible nondeterministic outcome of the action. In each such disjunction, all conjunctions should contain the same set of fluents.

The following is an example of an action with a nondeterministic outcome.

$$\begin{aligned} &T \Rightarrow \\ &[\,(Cause(s,FlipCoin,s',has\text{–}coin,F) \wedge \\ &\quad Cause(s,FlipCoin,s',heads\text{–}up,F)) \vee \\ &\quad (Cause(s,FlipCoin,s',has\text{–}coin,F) \wedge \\ &\quad Cause(s,FlipCoin,s',heads\text{–}up,T))\,] \end{aligned} \tag{13}$$

No-change (NCH). If a fluent is not explicitly influenced by an action (that is appears in a *Cause* statement) it is assumed not to change. This is realised with the no-change axiom (NCH), which only allows fluents that are occluded to change. NCH states that if s' is a situation resulting from doing a in s then only those fluents explicitly influenced by the action a (that is to say $Occl(f,a,s)$) may change. The fluents that are not occluded must have the same truth values after a as they had before a. NCH is as follows.

$$Res(s,a,s') \Rightarrow \forall f.[\neg Occl(f,a,s) \Rightarrow Holds(f,s) \equiv Holds(f,s')] \tag{14}$$

Possibility to execute actions (POSS and EXE). Rules for when actions can be executed (POSS) are written in the form $Poss(a, s) \Rightarrow \alpha(s)$. There is also a condition that s' is a resulting situation only if a can be executed in s (EXE).

$$Res(s, a, s') \Rightarrow Poss(a, s) \tag{15}$$

3.2 Reasoning with a theory

In order to achieve a theory suitable for reasoning, a step of nonmonotonic processing is required. In the two-doors scenario, the possibility conditions (POSS) and effect laws (LAW) for the two actions of going through the left door and the right door can be defined as follows.

$$\begin{aligned} &Poss(GoLeft, s) \Rightarrow \\ &Holds(alive, s) \wedge \neg Holds(free, s) \end{aligned} \tag{16}$$

$$\begin{aligned} &(Holds(lf, s) \Rightarrow \\ &\quad Cause(s, GoLeft, s', free, T)) \wedge \\ &(\neg Holds(lf, s) \Rightarrow \\ &\quad Cause(s, GoLeft, s', alive, F)) \end{aligned} \tag{17}$$

$$\begin{aligned} &Poss(GoRight, s) \Rightarrow \\ &Holds(alive, s) \wedge \neg Holds(free, s) \end{aligned} \tag{18}$$

$$\begin{aligned} &(Holds(lf, s) \Rightarrow \\ &\quad Cause(s, GoRight, s', alive, F)) \wedge \\ &(\neg Holds(lf, s) \Rightarrow \\ &\quad Cause(s, GoRight, s', free, T)) \end{aligned} \tag{19}$$

Using the definition of $Cause$ in (10), (11) on the statements (17) and (19) results in (20) and (21) respectively.

$$\begin{aligned} &Holds(lf, s) \Rightarrow [Occl(free, GoLeft, s) \wedge \\ &\quad (Res(s, GoLeft, s') \Rightarrow Holds(free, s'))] \wedge \\ &\neg Holds(lf, s) \Rightarrow [Occl(alive, GoLeft, s) \wedge \\ &\quad (Res(s, GoLeft, s') \Rightarrow \neg Holds(alive, s'))] \end{aligned} \tag{20}$$

$$\begin{aligned} &Holds(lf, s) \Rightarrow [Occl(alive, GoRight, s) \wedge \\ &\quad (Res(s, GoRight, s') \Rightarrow \neg Holds(alive, s'))] \wedge \\ &\neg Holds(lf, s) \Rightarrow [Occl(free, GoRight, s) \wedge \\ &\quad (Res(s, GoRight, s') \Rightarrow Holds(free, s'))] \end{aligned} \tag{21}$$

This in turn can be rewritten as follows, separating the *Occl* (22) and *Res* (23) parts.

$$
\begin{array}{l}
[\,(Holds(lf,s) \wedge \langle f,a\rangle \in \{\langle free, GoLeft\rangle, \\
\quad \langle alive, GoRight\rangle\}) \vee \\
\;(\neg Holds(lf,s) \wedge \langle f,a\rangle \in \{\langle alive, GoLeft\rangle, \\
\quad \langle free, GoRight\rangle\})\,] \Rightarrow Occl(f,a,s)
\end{array} \tag{22}
$$

$$
\begin{array}{l}
Res(s,a,s') \Rightarrow \\
\;[a = GoLeft \Rightarrow \\
\quad (Holds(lf,s) \Rightarrow Holds(free,s') \wedge \\
\quad\; \neg Holds(lf,s) \Rightarrow \neg Holds(alive,s'))] \wedge \\
\;[a = GoRight \Rightarrow \\
\quad (Holds(lf,s) \Rightarrow \neg Holds(alive,s') \wedge \\
\quad\; \neg Holds(lf,s) \Rightarrow Holds(free,s'))]
\end{array} \tag{23}
$$

The nonmonotonic step consists of turning the respective sums of conditions for *Poss*, *Occl* and *Res* into biconditionals. The conditions for *Poss* are all contained in POSS and give the following sentence, which characterizes exactly when specific actions can be executed.

$$
\begin{array}{l}
Poss(a,s) \equiv \\
\;((a = GoLeft \vee a = GoRight) \Rightarrow \\
\quad Holds(alive,s) \wedge \neg Holds(free,s))
\end{array} \tag{24}
$$

The *Occl* predicate should hold only when explicitly stated in LAW. Thus, the conditions in LAW for *Occl* (22) give the following sentence, which characterizes exactly under what conditions specific fluents may change.

$$
\begin{array}{l}
Occl(f,a,s) \equiv \\
(Holds(lf,s) \wedge \\
\langle f,a\rangle \in \{\langle free, GoLeft\rangle, \langle alive, GoRight\rangle\}) \vee \\
(\neg Holds(lf,s) \wedge \\
\langle f,a\rangle \in \{\langle alive, GoLeft\rangle, \langle free, GoRight\rangle\})
\end{array} \tag{25}
$$

Conditions for *Res* are contained in NCH (14), EXE (15) and LAW (23). Together they provide the following completion of *Res*.

$$
\begin{array}{l}
Res(s,a,s') \equiv \\
\quad [\ (\forall f.\neg Occl(f,a,s) \Rightarrow \\
\qquad (Holds(f,s) \equiv Holds(f,s')))\wedge \\
\qquad Poss(a,s)\wedge \\
\qquad [a = GoLeft \Rightarrow \\
\qquad\quad (Holds(lf,s) \Rightarrow Holds(free,s')\wedge \\
\qquad\quad \neg Holds(lf,s) \Rightarrow \neg Holds(alive,s'))]\wedge \\
\qquad [a = GoRight \Rightarrow \\
\qquad\quad (Holds(lf,s) \Rightarrow \neg Holds(alive,s')\wedge \\
\qquad\quad \neg Holds(lf,s) \Rightarrow Holds(free,s'))]\ ]
\end{array}
\tag{26}
$$

Observe that *Res* is determined by *Poss* and *Occl*. Informally, this means that first one decides when actions are possible to execute and in what circumstances fluents are allowed to change. Subsequently, and with these constraints in mind, the result relation is generated.

The completion of *Res* implies that the extension of *Res* is maximized. Why this is important becomes clear if one considers a nondeterministic action, such as the coin flipping action (13). From a situation s such that $Holds(has\text{–}coin, s)$, *FlipCoin* should be able to lead both to a situation s'_1 such that $\neg Holds(heads\text{–}up, s'_1)$ and to another situation s'_2 such that $Holds(heads\text{–}up, s'_2)$. That is to say, both $Res(s, FlipCoin, s'_1)$ and $Res(s, FlipCoin, s'_2)$ should hold. In order to guarantee this, *Res* is maximized. If *Res* was not maximized, there would be models where either $\neg Res(s, FlipCoin, s'_1)$ or $\neg Res(s, FlipCoin, s'_2)$ is true, and in those models *FlipCoin* would not be nondeterministic. Even worse, there might be models where both $\neg Res(s, FlipCoin, s'_1)$ and $\neg Res(s, FlipCoin, s'_2)$ are true, and thus *FlipCoin* would not be possible to execute. In short, maximizing *Res* implies that there is one resulting situation for each possible outcome of an action in a given situation.

The completions of the three predicates, that is (24), (25) and (26), contain all relevant information regarding the effects of actions, and can be used for theorem proving together with SE and UNA. Figure 2 shows one model. One can notice that there are only arcs starting from the upper two situations, as these are the only ones that satisfy *Poss* (24). Furthermore, observe that all arcs result in fluent changes that are sanctioned by *Occl* (25).

To summarize, the process of generating the completions of *Occl*, *Poss* and *Res* is as follows. (a) The extension of *Occl* is minimized relative to

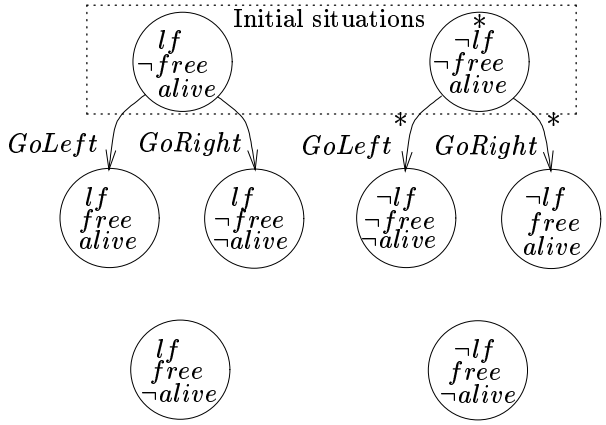

Figure 2: A model of the two-doors scenario (all other models are isomorphic). Neither $GoLeft$ nor $GoRight$ leads to the goal $free$ from all initial situations satisfying $\neg free \wedge alive$. However, if SE were excluded and the situation and arcs marked with asterisks were missing, $GoLeft$ would have appeared to lead to $free$.

LAW, where it only occurs positively. (b) The extension of $Poss$ is maximized relative to POSS, where it only occurs negatively. (c) The extension of Res is maximized relative to NCH, EXE and LAW, where it only occurs negatively. This process can be automated with second-order circumscription [82], as follows.

$$\text{UNA} \wedge \text{SE} \wedge Circ[\text{LAW}; Occl] \wedge Circ[\text{POSS}; \neg Poss] \wedge Circ[\text{NCH} \wedge \text{EXE} \wedge \text{LAW}; \neg Res] \tag{27}$$

The expression $Circ[\Gamma; \neg P]$ denotes that the extension of the predicate P is maximized. Although the result of this circumscription is a second-order formula, two results by Lifschitz [83] and the fact that $Occl$, $\neg Poss$ and $\neg Res$ only occur positively within the theories relative to which they are circumscribed imply that it is equivalent to a first-order formula. Thus, the circumscribed parts of the theory are completely first-order. However, this is not (always) the case for the situation existence axiom, or for the induction axiom mentioned in the next section.

4 Action composition

With the use of a Res predicate above, a sequence of actions is no longer represented as a term of nested do's. Instead, a type $\mathcal{N}$ for composite actions is introduced together with a predicate $Res^*(s, n, s')$. It is assumed that this type is supported by a (second-order) induction axiom (IND). Constructs for primitive actions and sequences are defined as follows.

$$Res^*(s, do(a), s') \equiv Res(s, a, s') \tag{28}$$

$$Res^*(s, seq(n_1, n_2), s') \equiv \\ \exists s''.Res^*(s, n_1, s'') \wedge Res^*(s'', n_2, s') \tag{29}$$

A $Poss^*$ predicate for composite actions can be defined in terms of Res^* and $Poss$.

$$Poss^*(do(a), s) \equiv Poss(a, s) \tag{30}$$

$$Poss^*(seq(n_1, n_2), s) \equiv [\, Poss^*(n_1, s) \wedge \\ \forall s'.Res^*(s, n_1, s') \Rightarrow Poss^*(n_2, s')\,] \tag{31}$$

Notice that this definition implies that all the primitive actions along any path described by $seq(n_1, n_2)$ must be possible to execute.

Now, the positive (negative) plan existence problem can be formulated as follows (let $I(s)$ describe the initial conditions).

$$(\neg)\exists n.\, \forall s.[I(s) \Rightarrow [Poss^*(n, s) \wedge \\ \forall s'.Res^*(s, n, s') \Rightarrow G(s')]] \tag{32}$$

For instance, consider the two-doors scenario again. It is obvious that for neither $n = do(GoLeft)$ nor $n = do(GoRight)$ (Fig. 2) does it hold that

$$\forall s.[Holds(alive, s) \wedge \neg Holds(free, s) \Rightarrow \\ [Poss^*(n, s) \wedge \\ \forall s'.Res^*(s, n, s') \Rightarrow Holds(free, s')]]. \tag{33}$$

Also observe the importance of the situation existence (SE) axiom. If there are models where situations are missing, one might fail to conclude that there is no plan (Fig 2).

When planning with an incompletely specified initial situation and non-determinism, the concept of a plan as a sequence appears to be insufficient. What one would like is the possibility to take different courses of actions depending on the circumstances. This is the approach taken in contingency

planning or conditional planning [110]. In order to demonstrate the feasibility of representing plans as complex terms, it is important to address the problem of conditional constructs. The following is a definition for Res^* and $Poss^*$ for a conditional construct (the predicate $Holds'$ denotes that a condition holds).

$$\begin{aligned} &Res^*(s, cond(c, n_1, n_2), s') \equiv \\ &\quad (Holds'(c, s) \wedge Res^*(s, n_1, s')) \vee \\ &\quad (\neg Holds'(c, s) \wedge Res^*(s, n_2, s')) \end{aligned} \tag{34}$$

$$\begin{aligned} &Poss^*(cond(c, n_1, n_2), s) \equiv \\ &\quad (Holds'(c, s) \wedge Poss^*(n_1, s)) \vee \\ &\quad (\neg Holds'(c, s) \wedge Poss^*(n_2, s)) \end{aligned} \tag{35}$$

In order to let one of the conditional branches be empty, a *noop* action is introduced.

$$Res^*(s, noop, s') \equiv s = s' \tag{36}$$

$$Poss^*(noop, s) \equiv T \tag{37}$$

The definition of *cond* raises a question regarding the status of the condition component c. One possibility is that c is a fluent, like in

$$cond(loaded, noop, do(Load)).$$

In [79], c is a sensing action that returns either 0 (for false) or 1 (for true):

$$cond(check\_loaded, noop, do(Load)).$$

A more general approach, following a proposal by McCarthy [94] (see also [105]), is to represent complex conditions as terms as follows. First, there is a type $\mathcal{D}$ for object designators, which are either used as variables x_i (short for $\mathsf{x}(i)$ where $i \in \mathbb{N}$), or as constants; the latter are formed with the function $@ : \mathcal{O} \to \mathcal{D}$. A function $V : \mathcal{D} \to \mathcal{O}$ maps object designators to objects. In particular, $V(@(x)) = x$. For instance, $@(fred)$ (or $@fred$) denotes a name that via V refers to the object $fred$. Then there is a type $\mathcal{C}$ for complex conditions. It includes references to the different fluents in the domain, like alive or $\mathsf{in}(@robot, @room1)$, and constructs equivalent to the standard logical connectives and quantifiers and equality, like $\overline{\vee} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ (or or to retain a term-like syntax). A predicate $Holds' : \mathcal{C} \times \mathcal{S}$ is used for complex conditions like $Holds$ is used for single fluents. $Holds'$ is defined in

terms of $Holds$. For instance, $Holds'(\mathsf{loaded} \nabla \mathsf{alive}, s) \equiv Holds(loaded, s) \vee Holds(alive, s)$.

The following axioms define the $Holds'$ predicate. For each fluent designator symbol f_i corresponding to a fluent symbol f_i, there is an axiom as follows.

$$Holds'(\mathsf{f}_i(d_1, \ldots, d_n), s) \equiv \\ Holds(f_i(V(d_1), \ldots, V(d_n)), s) \tag{38}$$

The reified equality relation $\overline{=}$ and the reified connectives $\overline{\neg}, \overline{\mathbb{C}} \in \{\overline{\wedge}, \overline{\vee}, \overline{\Rightarrow}, \overline{\equiv}\}$ and quantifiers $\overline{\mathbf{Q}} \in \{\overline{\exists}, \overline{\forall}\}$ are defined as follows.

$$\begin{aligned} &Holds'(d_1 \overline{=} d_2, s) \equiv V(d_1) = V(d_2) \\ &Holds'(\overline{\neg} c, s) \equiv \neg Holds'(c, s) \\ &Holds'(c_1 \, \overline{\mathbb{C}} \, c_2, s) \equiv \\ &\quad Holds'(c_1, s) \, \mathbb{C} \, Holds'(c_2, s) \\ &Holds'(\overline{\mathbf{Q}} \, v.c, \; s) \equiv \\ &\quad \mathbf{Q}x. \, Holds'(\mathsf{sb}(c, v, @(x)), s) \end{aligned} \tag{39}$$

The substitution functions $\mathsf{sb} : \mathcal{D} \times \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ and $\mathsf{sb} : \mathcal{C} \times \mathcal{D} \times \mathcal{D} \to \mathcal{C}$ are defined in the obvious way. Finally, a set of unique name axioms are supplied for the conditions. Essentially, two distinct terms of types $\mathcal{D}$ or $\mathcal{C}$ always denote two distinct objects, even if they refer to the same object via the V function or have the same truth value via the $Holds'$ predicate. Conditions and designators are considered syntactical entities, and equality is based on syntactical identity, not reference.

Now it is possible to construct conditional plans such as

$$n = cond(\mathsf{If}, do(GoLeft), do(GoRight)) \tag{40}$$

which is a solution to the two-doors problem. It is straightforward to show that n satisfies (33).

A final remark: the condition in the plan above refers to the actual state of the world. Preferably, such decisions should be based on the knowledge of the agent. This requires a theory that explicitly represents the agent's knowledge or beliefs [105, 79]. This is compatible with the technique presented in this section.

5 Conclusions

Situation Calculus is a formalism for reasoning about action and change that supports reasoning about alternative courses of action on the object

level. However, it fails to provide adequate object-level support for reasoning about the different ways the world can develop relative to a specific choice of actions. This results in anomalies when Situation Calculus is used for higher-level reasoning such as planning, as illustrated by the two-doors scenario. This paper has presented a modified version of SC that deals with this problem. It allows multiple initial situations and multiple alternative results of actions within individual models of an SC theory. In fact, the approach presented here can be considered a "modalization" of SC — it is possible to reason about possible and necessary results of a course of actions — although no explicit modal operators have been introduced. Three important technical considerations are the use of a *Res* relation instead of a *do* function (for nondeterminism, which is central to the paper), a situation existence axiom (as there are no terms denoting individual situations) and a type for composite actions, more suitable than individual situations for representing plans. Also observe the simple circumscription policy employed (27), which does not involve any prioritizing or varying of predicates. This policy is a variant of Sandewall's PMON policy [124], originally for an integer time structure, which has been altered to fit the new temporal structure.

The motivating example of this paper is analogous to an example from [92] with a monkey and two boxes, one of which contains a banana and one of which contains a bomb. In their SC-based approach, the synthesis of plans is supported by a deductive-tableau inference system. The point the authors make with the example is that the proofs for generating a plan should be constructive. Manna's and Waldinger's plan theory introduces plans as explicit objects. The application of an action or a plan to a state is modeled as a function yielding a new state, which implies determinism. Although their motives are similar, the approach of Manna and Waldinger is substantially different from the work presented in this paper; in particular, the authors are not concerned with axiomatic solutions to the frame problem.

There are also similarities to work on planning with sensory actions by (among others) Levesque [79] and Davis [22], which involve explicit theories of knowledge. Levesque introduces complex action terms with sequential and conditional composition. Furthermore, a plan is required to lead to the goal from all situations that are compatible with the agent's knowledge of the initial situation, and not just the initial situation. However, the purpose is not to deal with the problems addressed in this paper. In particular, actions are assumed to be deterministic. Actually, as Levesque uses a *do* function, the agent will "know" that actions are deterministic, even if they are specified not to be. Also Davis provides a type for plans, and in addition

a *Result* relation. The purpose of the relation is to let plans have unspecified results, and not to model nondeterminism. Davis mentions nondeterminism as a future problem.

Dynamic logic [116] is a polymodal formalism that has been applied to reasoning about action and change and to planning [122]. In dynamic logic semantics, a relation defines the possible next states when an action is executed in a state, in a manner similar to the $Res(s, a, s')$ relation in this paper.

To conclude, the essential novelty of this paper is that it combines plans as explicit objects with not only incomplete initial information but also nondeterministic effects. This is done in classical logic, with the aid of a relatively simple circumscriptive policy based on a compact yet powerful solution to the frame problem. Regarding future work, an interesting project would be to combine nondeterminism and knowledge operators.

Acknowledgments

I would like to thank Silvia Coradeschi, Patrick Doherty, Marcus Bjäreland and the anonymous referees for their comments on previous versions of this paper. This research has been supported by the Swedish Research Council for Engineering Sciences (TFR) grant 96-728.

Paper V

Anything Can Happen: on narratives and hypothetical reasoning

Lars Karlsson
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
Email: larka@ida.liu.se

Abstract

In this paper we consider the integration of three desirable properties in formalisms for action and change, namely representing and reasoning about (I) metric time, (II) alternative ways the world can develop relative to a specific choice of actions, and (III) alternative choices of actions. A language is presented that integrates these three aspects, and a translation to predicate calculus is provided. The central idea is to introduce narratives, that is chronological descriptions of what happens in the world over time, as objects which can be manipulated within the logic and to provide operators for reasoning about the properties of narratives.

1 Introduction

The purpose of this paper is to consider what it means to represent and reason about action structures such as sequences and narratives as objects

in a logical language. Representing action structures as objects enables reasoning hypothetically about different choices of actions and their results. More concretely, this paper presents a logic, called the narrative logic (NL for short) for reasoning about action structures called narratives. The core of NL, that is the representation of actions and their effects, stems directly from Sandewall's PMON logic [124, 26]. From there, NL inherits a metric time structure and thereby provides a rich representation of temporal relations. On top of this PMON core, NL provides reasoning about different narratives, and the different ways the world can develop for a specific narrative. This is achieved by reifying two important concepts in PMON, namely the concept of an action schedule (that is narrative) and the concept of a development (that is a sequence of states over time). The combination of these features — explicit time, support for reasoning about alternative narratives and about alternative developments of a narrative — is what makes NL a novel and interesting formalism, suitable as a foundation for theories about plans and explanations.

When reasoning about action and change, branching time is one way to view time and actions and how they relate. In situation calculus [97, 118] (SC for short), the predominant event-based branching formalism, time is seen as generated by the execution of actions. There is an initial situation (denoted S_0), and the execution of an action in a situation results in a new situation, which yields a branching temporal structure (Fig. 1 (a)). A situation is denoted by a term constructed from the corresponding action sequence, e.g. $result(Fire, result(Load, S_0))$. The strength of this view is that it supports reasoning about alternative action sequences. In particular, action sequences are first-order objects and can for instance be quantified over. Thus, one can make statements such as "if I load the gun and fire, the turkey will be dead": $\neg Holds(alive, result(Fire, result(Load, S_0)))$. Statements about situations in general are also possible. For instance, the fact that there is some sequence of actions (plan) that results in a situation where a condition (goal) G holds can be stated as $\exists s\,[G(s) \wedge ex(s)]$. Here, s is a situation variable and $ex(s)$ is a condition that the situation is reachable from the initial situation S_0. The weakness of SC is that there is no underlying, independent notion of time which actions can relate to. This makes it difficult to express nontrivial temporal properties of and relations between actions, such as partially ordered or overlapping action occurrences.[1]

[1] Although metric time can be added on top of the branching structure [112, 120], the underlying branching structure still enforces strong restrictions on how actions can be temporally related. In particular, the timing and ordering of actions needs to be complete.

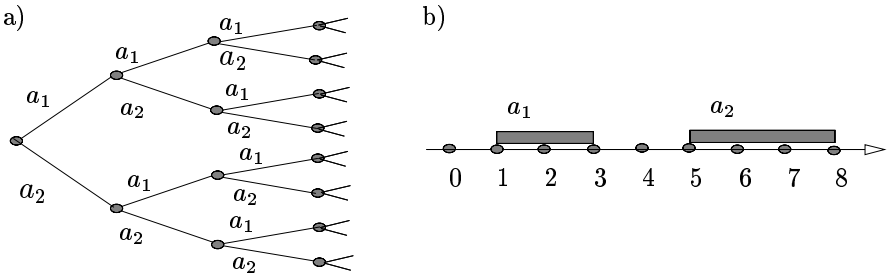

Figure 1: (a) Branching time structure formed by execution of actions where the nodes are situations; and (b) linear time structure and action occurrences.

A second way is to separate the notions of time and action, which is the case in narrative-based approaches [69, 6, 124]. Time is represented as a linear structure (for instance the integers), and the occurrence of an action is represented by connecting the action to the temporal structure with a special relation, like in $occurs(2, 5, Load)$ and $occurs(7, 9, Fire)$ (Fig. 1 (b)). The pros and cons of this approach are more or less complementary to the ones of situation calculus. Expressing temporal properties and relations is straightforward, but action structures (narratives) become metalogical objects (logical statements), and reasoning about alternative narratives has to be done in terms of reasoning about alternative theories or logical statements. For instance, the statement "if I load the gun and fire, the turkey will be dead" has to be formulated $\Gamma \wedge \alpha \models \neg Holds(10, alive)$ or $\Gamma \models (\alpha \Rightarrow \neg Holds(10, alive))$ where α specifies what actions occur ($Load$ and $Fire$) and Γ defines action laws (among other things). Thus, it is to some extent possible to reason hypothetically about narratives, but when it comes to general statements such as the plan existence formulation mentioned in connection with SC above, then one has either to use second-order quantification or to go outside the logic and state that "there is a consistent α of the form X such that $\Gamma \wedge \alpha \models G$".

Besides notions of time and actions, reasoning about action and change usually involves a notion of state, that is what facts are true or false at a specific point in time. This is commonly represented with a *holds* relation between fluents (temporally dependent properties or relations) and situations or time-points. The states over a certain branch or line of time can be considered to form a development. In figure 1, each branch is associated with one (partial) development, and the narrative with one development.

However, given a specific action sequence or narrative, the world may de-

velop in alternative ways due to different initial states and non-deterministic results of actions, and in the case of narratives, due to incomplete information regarding the timing and duration of actions. In many reasoning tasks it is important to be able to explicitly reason about these alternative developments of the world, for instance in plan synthesis (given observations of some initial state and a goal, find a sequence/narrative that necessarily leads to the goal) and explanation (given an observation of some state at a later time point, find a sequence/narrative that might have lead to that state). The keywords here are "necessarily" and "might". Neither the simple branching time nor linear time models presented above supports this form of reasoning, as there is only one development for each action sequence/narrative.[2] Consequently, there are problems with the aforementioned SC formulation of plan existence. It has been shown that if one introduces incomplete initial information or nondeterministic effects of actions, then this formulation sometimes states that there is a plan when actually there is none [62]. The root of the problem is that it is not possible to formulate a condition that the plan should work for all potential initial states and nondeterministic results; instead, the formulation says "for each different combination of initial conditions/nondeterministic outcomes, there is a plan".

In order to deal with the reasoning problems mentioned above appropriately on the object level, one has to view action sequences/narratives as entities having multiple parallel developments (Fig. 2). By quantifying over developments (either explicitly or with modal operators) one can distinguish between what must and what might be the result of a specific choice of actions. Dynamic logic [116] is a polymodal formalism with a branching temporal structure similar to the one of SC, which distinguishes between necessary and possible results of actions. Davis's [22] and Levesque's [79] versions of SC involve explicit theories of knowledge, and the knowledge operators there can be seen as "necessary"-operators. These two formalisms yield structures resembling Fig. 2 (a). Karlsson's SC version [62] does not have explicit modalities, but instead axiomatizes the existence of situations corresponding to each possible state in order to make both "necessary" and "possible" properties hold as intended. A noteworthy feature of the abovementioned SC versions is that they introduce a type for action structures (sequences, conditionals, etc.) which is separate from situations, and reasoning about alternative action choices is done in terms of this type. Each

[2] Naturally, the different logical models of a situation calculus or narrative-based theory might represent different world developments. However, in each model there is only one such development, and syntactically it is not possible to construct statements that refer to multiple parallel developments.

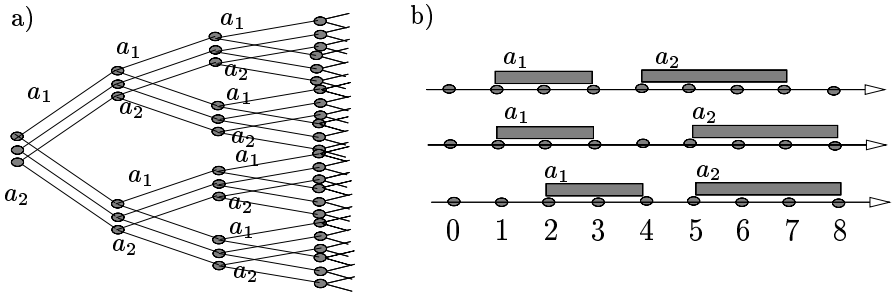

Figure 2: Branching (a) and linear (b) time structures with multiple parallel developments.

action structure can be considered to be mapped to a set of developments. With a type for action structures, whether the underlying temporal structure is branching or linear should no longer be of importance for the ability to reason about different choices of actions.

The narrative logic (NL) is a product of the insight that the key to reasoning about alternative choices of actions lies in the representation of action structures as objects and not in the underlying temporal structure, be it branching or linear. It is a formalism that has linear time and narratives as first-order objects associated with multiple developments. The fact that there is an independent notion of time enables a richer representation of temporal relations than is possible with event-based branching time. NL is based on the PMON entailment policy [124, 26] which has been proven correct for worlds with non-overlapping actions with context-dependent and non-deterministic effects. The central objects of NL are narratives, which have the following properties.

1. Narratives are terms in the logic, and can be quantified over and manipulated like other kinds of terms.

2. Narratives encode explicit assumptions or restrictions on what actions occur in the world.

3. These restrictions may be partial. For instance, the exact timing and duration of actions need not be specified.

4. These restrictions are local; typically, they concern specific intervals of time. Other types of locality, such as geographical locality, are possible but will not be considered in this paper.

5. Narratives can be incrementally extended to cover additional parts of the time line. Thus, narratives are dynamic entities, suitable for reasoning about alternative futures or pasts and for representing the beliefs of an agent in a dynamic environment.

6. Given a specific narrative, the world may develop in alternative ways due to incompleteness in the narrative (timing and duration of actions), different initial states and non-deterministic results of actions. Thus, a narrative corresponds to a set of developments, as in Fig. 2 (b).

The rest of the paper is organized as follows. Section 2 presents the syntax of NL and gives some examples. Section 3 shows how NL statements can be translated to predicate calculus and illustrates this with part of the examples from section 2. Section 4 presents some properties of narratives and and discusses applications, and section 5 presents some conclusions.

2 The surface language NL

In this section, the surface language NL is introduced. The different constructs of the language are presented together with their intuitions. In the next section, a translation from NL to predicate calculus (PC) is provided (where an induction axiom and four development existence axioms are second-order).

The basic types of NL are $\mathcal{F}$ for fluents, $\mathcal{A}$ for actions, $\mathcal{T}$ for time-points (non-negative integers) and $\mathcal{S}$ for temporal designators. The variable symbols f, a, t and s will be used for these types, respectively.

Narratives are constructed using narrative components that specify what actions occur, and temporal designators. The latter specify the time interval of the narrative component. Furthermore, they are used for temporal relations between actions. Note that temporal designators are of a type $\mathcal{S}$ which is different from the time-point type $\mathcal{T}$. This is due to point 3 in the introduction; narratives are partial restrictions, and permit variations of the timing of actions. If time-points ($\mathcal{T}$) had been used, the timing of all actions would have been totally fixed, and thus identical in all developments of the narrative. To provide a possibility to refer to fixed time-points in a narrative, there is a function @ that turns a time-point into a temporal designator.

Definition 15 *A temporal designator* $\mathbf{s} \in \mathcal{S}$ *is a term of the form* $\mathrm{S}n$, $n \in \mathbb{N}$ *(for unfixed designators),* $@\mathbf{t}$, $\mathbf{t} \in \mathcal{T}$ *(for fixed designators) or* $\mathbf{s}_1 + \mathbf{s}_2$ *(or*

alternatively to keep a term-like syntax: $\text{PLUS}(\mathbf{s}_1, \mathbf{s}_2)$*).*
An ordering constraint $c \in \mathcal{C}$ *is a term of the form* $(\mathbf{s} = \mathbf{s}')$, $(\mathbf{s} \leq \mathbf{s}')$, $(\mathbf{s} < \mathbf{s}')$, $(\neg\mathbf{c})$, $(\mathbf{c}_1 \wedge \mathbf{c}_2)$, $(\mathbf{c}_1 \vee \mathbf{c}_2)$ *or* T *for true (or alternatively:* $\text{EQ}(\mathbf{s}, \mathbf{s}')$ *etc.).*

Definition 16 *A narrative component* $\mathbf{n} \in \mathcal{N}$ *is either a variable* n *or a term of the form* $\text{D}(\mathbf{s}, \mathbf{s}', \mathbf{a})$ *(single action) or* $\text{J}[\mathbf{n}_1, \ldots, \mathbf{n}_k, \mathbf{c}]$ *where* $k \geq 0$ *(*J *stands for "join").*

The join operator combines subcomponents and relates them temporally (c is an ordering constraint).

Definition 17 *Let* $\mathbf{s}$, $\mathbf{s}'$ *be temporal designators,* $\mathbf{f}$ *a fluent,* $V \in \{\text{T}, \text{F}\}$ *and* $\mathbf{a}$ *an action. A holds statement is of the form* $\text{H}(\mathbf{s}, \mathbf{f})$*, a do statement is of the form* $\text{D}(\mathbf{s}, \mathbf{s}', \mathbf{a})$*, and a reassignment statement is of the form* $\text{R}(\mathbf{s}, \mathbf{s}', \mathbf{f}, V)$*. A temporal statement is of the form* $\mathbf{s} = \mathbf{s}'$, $\mathbf{s} \leq \mathbf{s}'$ *or* $\mathbf{s} < \mathbf{s}'$*.*

The H construct refers to the truth value of a particular fluent at a particular time point, and D to the occurrence of a particular action. The R construct denotes that the value of a fluent f is reassigned to the truth value V during the given interval $(\mathbf{s}, \mathbf{s}']$. It is used exclusively for specifying the effects of actions. Notice the overloading of the D symbol and the temporal relations. They are used both for terms and statements; however, it should always be clear from the context what is intended.

A development is characterized by what actions occur and what fluents hold over time, and how temporal designators are mapped to time-points. The Law construct is used for stating axioms regarding the effects of actions on fluents. These, together with the principle of persistence (fluents only change due to the effects of actions) and a restriction that action occurrences may not temporally overlap, determine the set of (well-behaved) developments.

Definition 18 *A law statement is of the form*

$$\mathsf{Law}(\ \forall s, s'.\text{D}(s, s', a) \Rightarrow \textstyle\bigwedge_i [\alpha_i^s \Rightarrow \bigvee_j (\bigwedge_k \text{R}(s, s', f_{ik}, V_{ijk})) \wedge \beta_{ijk}^{s,s'})]\ ) \qquad (1)$$

where α_i^s *is a boolean combination of holds statements at time* s *and* $\beta_{ijk}^{s,s'}$ *(optional) is a logical combination of temporal statements and holds statements referring to fluents* f_{ik} *which specifies what holds in the interval* (s, s') *and the length of this interval.*

Example: The following are laws for a loading and a firing action.

$$\mathsf{Law}(\ \forall s, s'.\textsc{d}(s,s',\textsc{Load}) \Rightarrow \textsc{r}(s,s',\textsc{loaded},\textsc{T})\ ) \tag{2}$$

$$\begin{aligned}\mathsf{Law}(\ \forall s, s'.\textsc{d}(s,s',\textsc{Fire}) \Rightarrow\ (\ &\textsc{h}(s,\textsc{loaded}) \Rightarrow \\ &\textsc{r}(s,s',\textsc{loaded},\textsc{F}) \wedge \textsc{r}(s,s',\textsc{alive},\textsc{F})\ )\ )\end{aligned} \tag{3}$$

These general laws are then used for reasoning about specific narratives. Relative to a narrative, the world may develop in many different ways (see point 6 in the introduction). Therefore, NL has two operators of a modal character.

Definition 19 *A narrative consequence statement is of the form*

$$\mathsf{Nec}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \alpha\ )$$

(necessary consequence) or

$$\mathsf{Pos}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \alpha\ )$$

(possible consequence) where $\mathbf{s}$, $\mathbf{s}'$ *are temporal designators,* $\mathbf{n}$ *is a narrative component, and* α *is a logical combination of holds, do, temporal, and narrative consequence statements.*

Essentially, $\mathsf{Nec}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \alpha\ )$ states that for each development where the set of actions occurring within $[\mathbf{s}, \mathbf{s}']$ are exactly those in n and nothing is assumed about what occurs outside $[\mathbf{s}, \mathbf{s}']$, it is the case that α holds. That is to say, α is an inevitable consequence of the action occurrences in the narrative component $\mathbf{n}$. A formal definition is given in section 3.2. $\mathsf{Pos}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \alpha\ )$ is defined as $\neg\mathsf{Nec}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \neg\alpha\ )$.

Example: Given the laws (2), (3) above, the statements (4), (5) below are true. They state that if the gun is first loaded and then fired and no other action occurs in the interval [@0, S4], then it is a necessary consequence that the unfortunate turkey that is the target will die ($\neg\textsc{h}(\text{S4}, \textsc{alive})$ holds in all developments), but if only the fire action occurs, then it is possible that the turkey will survive ($\textsc{h}(\text{S3}, \textsc{alive})$ holds in some developments).

$$\begin{aligned}\mathsf{Nec}(\ @0,\ \text{S4},\ \textsc{j}[&\textsc{d}(\text{S1},\text{S2},\textsc{load}), \textsc{d}(\text{S3},\text{S4},\textsc{fire}), \\ &(@0 < \text{S1}) \wedge (\text{S2} \leq \text{S3})],\ \neg\textsc{h}(\text{S4},\ \textsc{alive})\ )\end{aligned} \tag{4}$$

$$\mathsf{Pos}(\ @0,\ \text{S3},\ \textsc{j}[\textsc{d}(\text{S1},\text{S2},\textsc{fire}), (@0 < \text{S1}) \wedge (\text{S2} \leq \text{S3})],\ \textsc{h}(\text{S3},\ \textsc{alive})\ ) \tag{5}$$

Informally, the truth of these statements can be justified as follows. Regarding (4), $\textsc{d}(\text{S1},\text{S2},\textsc{load})$ and (2) imply $\textsc{h}(\text{S2},\textsc{loaded})$, and the principle

of persistence implies H(S3,LOADED). Then, D(S3,S4,FIRE) and (3) imply ¬H(S4,ALIVE). Regarding (5), assume a development were ¬H(@0,LOADED)∧ H(@0,ALIVE) is true. By persistence, ¬H(S1,LOADED) is the case, and therefore D(S1,S2,FIRE) has no effects according to (3). So, by persistence H(S3,ALIVE) holds in this particular development. Therefore, H(S3,ALIVE) is a possible consequence.

Nec and Pos operators can also be nested, and nesting implies a gradual extension of a narrative (point 4 in the introduction). To make nesting of Nec and Pos operators meaningful, we need to refine the semantics of the Nec operator given above. For this purpose, we introduce the concept of a partial development, that is an equivalence class of developments that are equal within some given time segments. As the restrictions imposed by a narrative component on developments only concern a segment of time and nothing is assumed outside this segment, a narrative component can be seen as specifying a set of partial developments that are defined only over the associated time segment. Further, if the narrative component is applied to an existing partial development, then the narrative component can be considered to extend that partial development to also include the associated temporal segment in a way that satisfies the restrictions encoded in the narrative component. Often, a partial development can be extended in more than one way, so an extension results in a set of new partial developments.

Consequently, nesting of Nec and Pos operators implies a gradual extension of a set of partial developments. One starts with a set containing only the empty partial development (that is not defined for any temporal segment), then applies the narrative of the outermost operator which will result in a set of partial developments defined for the time segment of the outermost operator, and then applies the next operator to that set, and so on.

Based on the concept of a partial development, we can now give the following (informal) semantics for the Nec operator applied in the context of a partial development. *The expression* Nec(s, s', n, α) *states that for each new partial development that can be obtained by extending the given partial development over* $[s, s']$ *in such a way that the set of actions occurring within* $[s, s']$ *are exactly those in* n*, it is the case that the statement* α *holds.* For Pos, just replace "for each" with "for some" above.

Example: Consider the following statement.

$$\begin{array}{l}\mathsf{Nec}(\ @0,\ \text{S2},\ \text{J}[\text{D}(\text{S1,S2,LOAD}),\ (@0{\leq}\text{S1})], \\ \quad \mathsf{Nec}(\ \text{S2},\ \text{S4},\ \text{J}[\text{D}(\text{S3,S4,FIRE}),(\text{S2}{\leq}\text{S3})], \\ \quad\quad \neg\text{H}(\text{S4},\ \text{ALIVE})\ )\ )\end{array} \tag{6}$$

Eq. 6 states that whenever the loading action is executed, proceeding by executing the firing action inevitably results in the turkey's demise. That is to say, any partial development where the gun is loaded can only be extended with the gun being fired in such a way that the turkey dies. Note the structure of the statement: the last argument of the outer Nec consists of the inner Nec statement, which in turn has ¬H(S4, ALIVE) as its last argument.

Informally, the truth of (6) can be justified as follows. Take any partial development of the outermost narrative. The only action occurrence in the temporal segment [@0,S2] is D(S1,S2,LOAD). From D(S1,S2,LOAD) and (2) follows H(S2,LOADED). Now extend that partial development according to the inner narrative. As H(S2,LOADED) already holds, it follows from persistence that H(S3,LOADED) is true. Then, D(S3,S4,FIRE) and (3) imply ¬H(S4,ALIVE).

The Nec and Pos operators can be mixed with each other, and logically combined with holds, do and temporal statements. Finally, consistency of a narrative, i.e. the narrative corresponds to some possible development of the world, can be stated as Pos($s, s', n,$ T).[3]

Observe that there is a large degree of freedom in the temporal relations of a narrative component. It is easy to construct narrative consequence statements Nec(s, s', n, α) where actions in the narrative component n are not constrained to occur inside the given time segment $[s, s']$. The Within operator characterizes those components that actually do have their actions within the assigned time segment.

Definition 20 *A narrative verification statement is of the form*

$$\mathsf{Within}(s, s', n)$$

(all actions in n start and end within the interval defined by s, s').

The concept of a theory is defined as follows.

Definition 21 *An NL theory Γ_L is a finite set of law statements.*

Finally, $\models_{NL}$ denotes the consequence relation between an NL theory and a logical combination of narrative consequence (Nec and Pos) and verification statements α. The definition of $\models_{NL}$ is presented in section 3, and is based on a translation from NL to predicate calculus. In section 4, we return to properties and applications of the formalism.

[3]Note that narrative consistency is a property of objects in an NL theory, and not of the NL theory itself. Yet there is a strong analogy: logical consistency means that a theory has models, and narrative consistency means that a narrative has developments.

3 A translation to PC

In this section, a translation from NL to second-order predicate calculus is outlined. This implies that both a proof theory and a semantics are indirectly provided. The concept of a development, that is to say a precise description of what happens and holds in the world over time, plays a central role. The elements of this translation are as follows.

1. Narratives and developments are defined as first-order objects in PC.

2. A specification of the set of well-behaved developments is given in PC. A well-behaved development is one that obeys the action laws and the principle of persistence.

3. Next, the relation between narratives and developments is specified in PC. The concept of a narrative describing a development is defined.

4. A function for translating narrative logic statements to PC statements is defined.

5. Finally, a consequence relation is defined for NL based on this translation.

Points 1-3 constitute the base theory Γ_B of NL, and to this the translation of an NL theory Γ_L is added (point 4).

3.1 Base theory

The base theory Γ_B consists of the axioms presented below, and unique names and (optionally) domain closure axioms.

Narrative

A narrative is basically a sequence of interval- and narrative component pairs.

Definition 22 *A narrative* $\mathbf{m} \in \mathcal{M}$ *is a term of the form* ϵ *(for the empty narrative) or* $\mathrm{E}(\mathbf{m}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{n})$ *(the narrative* $\mathbf{m}$ *extended with the component* $\mathbf{n}$ *over the interval* $[\mathbf{s}_1, \mathbf{s}_2]$*).*

There is an induction axiom over narrative components (we utilize the fact that any join $\textsc{j}[\mathbf{n}_1, \ldots, \mathbf{n}_n, \mathbf{c}]$ can be rewritten $\textsc{j}[\mathbf{n}_1, \textsc{j}[\ldots, \textsc{t}], \mathbf{c}]$). $\textsc{j}[\textsc{t}]$ is the empty narrative component.

$$\begin{aligned}&\forall P[(\forall s, s', a[P(\textsc{d}(s, s', a))] \wedge P(\textsc{j}[\textsc{t}]) \wedge \\ &\quad \forall n, n', c[P(n) \wedge P(n') \Rightarrow P(\textsc{j}[n, n', c])]) \Rightarrow \\ &\forall n[P(n)]]\end{aligned} \tag{7}$$

Developments

Developments specify what actions occur and what fluents are true or false over time. Developments have the status of composite first-order objects in the PC translation, and the four components of a development define what actions occur, what fluents are true, what fluents might change, and how temporal designators are mapped to the time line in that particular development.

Definition 23 *A development $\delta \in \Delta$ is a tuple $\langle \mathbf{d}, \mathbf{h}, \mathbf{o}, \mathbf{v} \rangle \in (\mathcal{D} \times \mathcal{H} \times \mathcal{O} \times \mathcal{V})$ where*

- *the relation $D(\mathbf{d}, \mathbf{t}, \mathbf{t}', \mathbf{a})$ characterizes what actions occur;*
- *the relation $H(\mathbf{h}, \mathbf{t}, \mathbf{f})$ characterizes what fluents hold;*
- *the relation $X(\mathbf{o}, \mathbf{t}, \mathbf{f})$ characterizes what fluents might change due to reassignment;*
- *the function $V(\mathbf{v}, \mathbf{s})$ characterizes the values of designators $\mathbf{s}$. The co-domain of V is $\mathcal{T}$.*

When referring to the individual $\mathbf{d}$, $\mathbf{h}$, $\mathbf{o}$ and $\mathbf{v}$ components of a development δ, the notation δ^d, δ^h, δ^o and δ^v is used. Thus, the fact that an action occurs in the development δ is written $D(\delta^d, t, t', a)$, and that a fluent holds and is occluded is written $H(\delta^h, t, f)$ and $X(\delta^o, t, f)$, respectively. Finally, the value of a temporal designator in δ is obtained from $V(\delta^v, s)$.

Four second-order development existence axioms define the set of developments.[4] They state that for each possible instantiation Φ of the D predicate, there is a corresponding $\mathbf{d} \in \mathcal{D}$ and so on for H, X and V.

$$\forall \Phi \exists d \forall t, t', a[\Phi(t, t'a) \equiv D(d, t, t', a)] \tag{8}$$

[4] A parallel can be found in Baker's second-order situation existence axiom [9]. This axiom, used for obtaining a correct minimization of change, states that for each possible truth assignment for fluents, there is a situation where *Holds* realizes this assignment.

$$\forall\Phi\exists h\forall t, f[\Phi(t,f) \equiv H(h,t,f)] \tag{9}$$

$$\forall\Phi\exists o\forall t, f[\Phi(t,f) \equiv X(o,t,f)] \tag{10}$$

$$\begin{aligned}\forall\Phi[\ &(\forall t[\Phi(@(t)) = t] \wedge \forall s, s'[\Phi(s+s') = \Phi(s) + \Phi(s')]) \equiv \\ &\exists v[\forall s[\Phi(v,s) = V(v,s)]]\ ]\end{aligned} \tag{11}$$

Observe that (11) implies that $\forall v, t[V(v, @(t)) = t]$, that is to say @ has the intended meaning. [5]

Well-behaved developments

Each development tuple $\delta = \langle \mathbf{d}, \mathbf{h}, \mathbf{o}, \mathbf{v} \rangle$ represents a specific instantiation of the H, D and X relations and the V function. However, only a subset of these combinations represent well-behaved developments; namely those that satisfy the action laws and the principle of persistence. The predicate Wb determines this subset. Wb is a first-order version of the PMON entailment policy [124, 26]. In short, Wb states that a development should (a) satisfy the action laws (Law is generated from the Law statements; see 3.2), (b) have all action occurrences sequentially ordered, (c) have a minimal extension of occlusion (X), and (d) change may only occur at occluded timepoint-fluent pairs ($\otimes$ is exclusive or). It is points (c) and (d) that formalize the principle of persistence. X occurs in the translation of reassignment statements which are used for the effects of actions, and points (c) and (d) simply restrict the change of fluents to those cases when the fluent actually is influenced by an action. We assume the convention that free variables (such as δ below) are always universally quantified over.

$$\begin{aligned}&Wb(\delta) \equiv \{ \\ &\text{(a)}\ Law(\delta) \wedge \\ &\text{(b)}\ Seq(\delta^d) \wedge \\ &\text{(c)}\ \neg\exists o'\ [\forall t, f[X(o',t,f) \Rightarrow X(\delta^o,t,f)] \wedge \\ &\qquad\qquad \neg\forall t, f[X(\delta^o,t,f) \Rightarrow X(o',t,f)] \wedge \\ &\qquad\qquad Law(\langle \delta^d, \delta^h, o', \delta^v \rangle)\ ] \wedge \\ &\text{(d)}\ \forall f, t[\ H(\delta^h,t,f) \otimes H(\delta^h,t{+}1,f) \Rightarrow X(\delta^o,t{+}1,f)]\}\end{aligned} \tag{12}$$

[5] In addition, notice that the relation/function associated with a specific development component can be encoded as a set of pairs of natural numbers if the sets of actions, fluents and designators are countable. Thus, the cardinalities of the different sets of (equivalence classes of) development components are $2^{\aleph_0}$, the same as for the real numbers.

The sequentiality condition is defined as follows.

$$
\begin{array}{l}
Seq(d) \equiv \\
\forall t_1, t_1', a_1, t_2, t_2', a_2 [\ (D(d, t_1, t_1', a_1) \Rightarrow t_1 < t_1') \wedge \\
\qquad\qquad (D(d, t_1, t_1', a_1) \wedge D(d, t_2, t_2', a_2) \Rightarrow \\
\qquad\qquad\ (t_1' \leq t_2 \vee t_2' \leq t_1 \vee \\
\qquad\qquad\ \langle t_1, t_1', a_1 \rangle = \langle t_2, t_2', a_2 \rangle))\ ]
\end{array}
\tag{13}
$$

Relating narratives to developments

A narrative component describes a set of developments, namely those well-behaved developments that contain the same actions as the narrative component in the associated time segment. A predicate De formalizes this relation. It requires that all ordering constraints in the narrative component hold (Cn) and that actions occur in the assigned interval if and only if they are present in the narrative component (In).

$$
\begin{array}{l}
De(s, s', n, \delta) \equiv \\
\{Cn(n, \delta) \wedge V(\delta^v, s) \leq V(\delta^v, s') \wedge \\
\ \forall t, t', a [\ (V(\delta^v, s) \leq t < V(\delta^v, s') \vee \\
\qquad\ V(\delta^v, s) < t' \leq V(\delta^v, s')) \Rightarrow \\
\qquad\ (D(\delta^d, t, t', a) \ \equiv \ In(\delta, t, t', a, n))]\ \}
\end{array}
\tag{14}
$$

The In predicate formalizes the presence of an action inside a narrative component. Note that the action a is associated with two time-points t, t' which are matched against the temporal designators in the narrative component.

$$
\begin{array}{l}
In(\delta, t, t', a, \mathrm{D}(s, s', a)) \equiv \\
\quad (\langle t, t', a \rangle = \langle V(\delta^v, s), V(\delta^v, s'), a \rangle)
\end{array}
\tag{15}
$$

$$
\begin{array}{l}
In(\delta, t, t', a, \mathrm{J}[n_1, \ldots, n_k, c]) \equiv \\
\quad (\bigvee_i In(\delta, t, t', a, n_i))
\end{array}
\tag{16}
$$

The Cn predicates state that the temporal constraints specified in a narrative component/ordering constraint hold. Let $\mathbb{C} \in \{\wedge, \vee\}$ and $\mathbb{R} \in \{=, \leq, <\}$.

$$Cn(\delta, \mathrm{J}[n_1, \ldots, n_k, c]) \equiv (\wedge_i Cn(\delta, n_i)) \wedge Cn(\delta, c) \tag{17}$$

$$Cn(\delta, \mathrm{D}(s, s', a)) \equiv [V(\delta^v, s) < V(\delta^v, s')] \tag{18}$$

$$Cn(\delta, \mathrm{T}) \equiv T \tag{19}$$

$$Cn(\delta, (\neg c)) \equiv [\neg Cn(\delta, c)] \tag{20}$$

$$Cn(\delta, (c_1 \mathbb{C} c_2)) \equiv [Cn(\delta, c_1) \mathbb{C} Cn(\delta, c_2)] \tag{21}$$

$$Cn(\delta, (s_1 \mathbb{R} s_2)) \equiv [V(\delta^v, s_1) \mathbb{R} V(\delta^v, s_2)] \tag{22}$$

Example: The narrative component in (4) yields the following:

$$
\begin{aligned}
&De(@0,\ \text{s4},\ \text{J}[\text{D}(\text{s1},\text{s2},\text{LOAD}),\ \text{D}(\text{s3},\text{s4},\text{FIRE}),\\
&\qquad (@0{<}\text{s1})\wedge(\text{s2}{\leq}\text{s3})],\ \delta) \quad \equiv\\
&\{\,0 \leq V(\delta^v,\text{s4})\wedge\\
&\ (0 < V(\delta^v,\text{s1}) \wedge V(\delta^v,\text{s2}) \leq V(\delta^v,\text{s3})\wedge\\
&\ V(\delta^v,\text{s1}) < V(\delta^v,\text{s2}) \wedge V(\delta^v,\text{s3}) < V(\delta^v,\text{s4}))\wedge\\
&\ \forall t,t',a[(\ 0 \leq t < V(\delta^v,\text{s4}) \vee V(\delta^v,0) < t' \leq V(\delta^v,\text{s4})) \Rightarrow\\
&\ \ [D(\delta^d,t,t',a) \equiv\\
&\ \ \ (\langle t,t',a\rangle = \langle V(\delta^v,\text{s1}), V(\delta^v,\text{s2}), \text{LOAD}\rangle \vee\\
&\ \ \ \ \langle t,t',a\rangle = \langle V(\delta^v,\text{s3}), V(\delta^v,\text{s4}), \text{FIRE}\rangle)\,]\,]\,\}
\end{aligned}
\tag{23}
$$

Finally, there is a predicate Eq that represents that two developments are equal in the time intervals covered by a narrative. Recall the discussion about partial developments in section 2. A partial development is an equivalence class of complete developments that are equal within some temporal segments. Eq defines when two complete developments belong to the same equivalence class given a specific narrative.

$$
Eq(\delta_1,\delta_2,\epsilon) \tag{24}
$$

$$
\begin{aligned}
&Eq(\delta_1,\delta_2,\text{E}(m,s,s',n)) \equiv \{\\
&\ V(\delta_1^v,s) = V(\delta_2^v,s) \wedge V(\delta_1^v,s') = V(\delta_2^v,s')\wedge\\
&\ Eq(\delta_1,\delta_2,V(\delta_1^v,s),V(\delta_1^v,s'))\wedge\\
&\ Eq(\delta_1,\delta_2,m)\,\}
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
&Eq(\delta_1,\delta_2,t_1,t_2) \equiv \{\\
&\forall t,t',a[t_1 \leq t < t' \leq t_2 \Rightarrow\\
&(\,D(\delta_1^d,t,t',a) \equiv D(\delta_2^d,t,t',a)\wedge\\
&\ \ \exists t''D(\delta_1^d,t,t'',a) \equiv \exists t''D(\delta_2^d,t,t'',a))\wedge\\
&\ \ \exists t''D(\delta_1^d,t'',t',a) \equiv \exists t''D(\delta_2^d,t'',t',a)\,)\,]\wedge\\
&\forall t'[\ t_1 \leq t' \leq t_2 \Rightarrow\\
&\qquad \forall f[H(\delta_1^h,t',f) \equiv H(\delta_2^h,t',f)]\wedge\\
&\qquad \forall f[X(\delta_1^o,t',f) \equiv X(\delta_2^o,t',f)]\wedge\\
&\qquad \forall s[V(\delta_1^v,s) = t' \equiv V(\delta_2^v,s) = t']\,]\,\}
\end{aligned}
\tag{26}
$$

3.2 Translation function

This subsection presents the translation $[\![\cdot]\!]_m^\delta$ from NL statements to predicate logic statements. The superscript δ and subscript m are function arguments that refer to the development and narrative expressions that form

the context of the translation. The logical connectives and quantifiers are translated in the obvious way. Whenever a new variable is introduced, it is assumed to be fresh.

Basic statements

The translation from basic NL statements to predicate logic statements is straightforward: H to H, D to D, R to H and X, and temporal designators to V, as follows ($\mathbb{R} \in \{=, \leq, <\}$).

$$[\![\mathbf{s}_1 \mathbb{R} \mathbf{s}_2]\!]^{\delta}_{m} = V(\delta, \mathbf{s}_1) \mathbb{R} V(\delta, \mathbf{s}_2) \tag{27}$$

$$[\![\mathrm{H}(\mathbf{s}, \mathbf{f})]\!]^{\delta}_{m} = H(\delta^h, V(\delta^v, \mathbf{s}), \mathbf{f}) \tag{28}$$

$$[\![\mathrm{D}(\mathbf{s}, \mathbf{s}', \mathbf{a})]\!]^{\delta}_{m} = D(\delta^d, V(\delta^v, \mathbf{s}), V(\delta^v, \mathbf{s}'), \mathbf{a}) \tag{29}$$

$$\begin{aligned}&[\![\mathrm{R}(\mathbf{s}, \mathbf{s}', \mathbf{f}, \mathrm{T})]\!]^{\delta}_{m} = \\ &\quad (H(\delta^h, V(\delta^v, \mathbf{s}'), \mathbf{f}) \wedge \\ &\quad\ \forall t[V(\delta^v, \mathbf{s}) < t \leq V(\delta^v, \mathbf{s}') \Rightarrow X(\delta^o, t, \mathbf{f})])\end{aligned} \tag{30}$$

$$\begin{aligned}&[\![\mathrm{R}(\mathbf{s}, \mathbf{s}', \mathbf{f}, \mathrm{F})]\!]^{\delta}_{m} = \\ &\quad (\neg H(\delta^h, V(\delta^v, \mathbf{s}'), \mathbf{f}) \wedge \\ &\quad\ \forall t[V(\delta^v, \mathbf{s}) < t \leq V(\delta^v, \mathbf{s}') \Rightarrow X(\delta^o, t, \mathbf{f})])\end{aligned} \tag{31}$$

The translation of R is of special interest. Reassignment implies that the fluent $\mathbf{f}$ is true/false at the end of the interval $(\mathbf{s}, \mathbf{s}']$, *and* that $\mathbf{f}$ is not subject to persistence within the interval $(\mathbf{s}, \mathbf{s}']$.

Narrative consequence statements

The translation of the Nec operator is as follows; Pos($\mathbf{s}, \mathbf{s}', \mathbf{n}, \alpha$) is defined as $\neg$Nec($\mathbf{s}, \mathbf{s}', \mathbf{n}, \neg\alpha$).

$$\begin{aligned}&[\![\mathsf{Nec}(\ \mathbf{s}, \mathbf{s}', \mathbf{n},\ \alpha\ )]\!]^{\delta}_{m} = \\ &\forall \delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(\mathbf{s}, \mathbf{s}', \mathbf{n}, \delta')) \Rightarrow [\![\alpha]\!]^{\delta'}_{\mathrm{E}(m, \mathbf{s}, \mathbf{s}', \mathbf{n})}]\end{aligned} \tag{32}$$

This definition states that for each development δ' such that (a) δ' is equal to δ in the temporal segments in m, (b) δ' is well-behaved, and (c) the set of actions occurring within $[s, s']$ in δ' are exactly those in n and the ordering constraints in n are satisfied, the translation of α in the context of δ' and $\mathrm{E}(m, s, s', n)$ must hold. Compare this to the informal definition of Nec that was given in section 2. Condition (a) above corresponds to the condition

in the informal definition that the new partial development should be an extension of the given partial development.

Example: The translation of (4) is as follows, where **n** is the narrative component in question.

$$\begin{aligned}&\forall\delta,\delta_2[Eq(\delta,\delta_2,\epsilon)\wedge Wb(\delta_2)\wedge De(@0,\mathrm{S4},\mathbf{n},\delta_2)\Rightarrow\\&\quad\neg H(\delta_2^h,V(\delta_2^v,\mathrm{S4}),\textsc{alive})]\end{aligned}\tag{33}$$

Narrative verification statements

The Within operator is defined as follows.

$$\begin{aligned}&[\![\mathsf{Within}(\mathbf{s},\mathbf{s}',\mathbf{n})]\!]_m^\delta=\\&\forall\delta_2[(Eq(\delta,\delta_2,m)\wedge Wb(\delta_2)\wedge Cn(\delta_2,\mathbf{n}))\Rightarrow\\&\quad\forall t,t',a[\ In(\delta_2,t,t',a,\mathbf{n})\Rightarrow\\&\qquad V(\delta_2^v,\mathbf{s})\le t<t'\le V(\delta_2^v,\mathbf{s}')]]\end{aligned}\tag{34}$$

Law statements and NL theories

For an NL theory $\Gamma_L=\{\mathsf{Law}(\ \alpha_1\ ),\ldots,\mathsf{Law}(\ \alpha_k\ )\}$, the translation function is defined as follows:

$$[\![\Gamma_L]\!]_m^\delta=(Law(\delta)\equiv(\textstyle\bigwedge_{i=1}^k[\![\alpha_i]\!]_m^\delta))\tag{35}$$

Example: The laws in (2), (3) yield the following definition of Law:

$$\begin{aligned}&Law(\delta)\equiv\forall s,s'[\\&\quad D(\delta^d,V(\delta^v,s),V(\delta^v,s'),\textsc{load})\Rightarrow\\&\qquad(H(\delta^h,V(\delta^v,s'),\textsc{loaded})\wedge\\&\qquad\forall t[V(\delta^v,s)<t\le V(\delta^v,s')\Rightarrow\\&\qquad\quad X(\delta^o,t,\textsc{loaded}))]\wedge\\&\quad D(\delta^d,V(\delta^v,s),V(\delta^v,s'),\textsc{fire})\Rightarrow\\&\qquad H(\delta^h,V(\delta^v,s),\textsc{loaded})\Rightarrow\\&\qquad(\neg H(\delta^h,V(\delta^v,s'),\textsc{alive})\wedge\\&\qquad\forall t[V(\delta^v,s)<t\le V(\delta^v,s')\Rightarrow X(\delta^o,t,\textsc{alive})]\wedge\\&\qquad\neg H(\delta^h,V(\delta^v,s'),\textsc{loaded})\wedge\\&\qquad\forall t[V(\delta^v,s)<t\le V(\delta^v,s')\Rightarrow X(\delta^o,t,\textsc{loaded})]\ )\ ]\end{aligned}\tag{36}$$

The following theorem makes it possible to obtain a more convenient formulation of what is occluded than the statement about minimal occlusion provided in the definition of Wb (12(c)).

Theorem 2 *Let $\Gamma = \Gamma_B \wedge [\![\Gamma_L]\!]$ be a translated theory. Then the definition of Wb (12) is logically equivalent to a sentence*

$$
\begin{array}{l}
Wb(\delta) \equiv \{ \\
\text{(a)}\ Law(\delta) \wedge \\
\text{(b)}\ Seq(\delta^d) \wedge \\
\text{(c)}\ \forall f, t[X(\delta^o, t, f) \equiv \Lambda(\delta, t, f)] \wedge \\
\text{(d)}\ \forall f, t[\ H(\delta^h, t, f) \otimes > H(\delta^h, t{+}1, f) \Rightarrow \\
\qquad\qquad X(\delta^o, t{+}1, f)]\}
\end{array}
\tag{37}
$$

where $\Lambda(\delta, t, f)$ is defined entirely in terms of D, H and comparisons between temporal, fluent and action terms.

Proof (sketch). In action laws, R only occurs positively (1), and due to the translation of R (30), (31), X will only occur positively inside the definition of Law that results from the translation (35). Thus, it is possible to reformulate (35) after translation as $\forall \delta, m[Law(\delta) \equiv (\forall f, t[\Lambda(\delta, t, f) \Rightarrow X(\delta^o, t, f)] \wedge B)]$, where B constitutes the occlusion-free parts of the laws. Thus, if $\Lambda(\delta, t, f) \equiv X(\delta^o, t, f)$ then δ^o is minimal; there can be no o' that satisfies the condition in (12(c)) as any such o' that is smaller than δ^o would fail to satisfy $\Lambda(\delta, t, f) \Rightarrow X(o', t, f)$.

Example: The action laws in (36) yield the following definition of X.

$$
\begin{array}{l}
\forall f, t[X(\delta^o, t, f) \equiv \exists s, s', a[\\
\quad V(\delta^v, s) < t \leq V(\delta^v, s') \wedge \\
\quad D(\delta^d, V(\delta^v, s), V(\delta^v, s'), a) \wedge \\
\quad (\ (a = \textsc{load} \wedge f \in \{\textsc{loaded}\}) \vee \\
\quad\ (a = \textsc{fire} \wedge H(\delta^h, V(\delta^v, s), \textsc{loaded}) \wedge \\
\quad\ f \in \{\textsc{loaded}, \textsc{alive}\})\ )\ ]\ ]
\end{array}
\tag{38}
$$

Consequence relation

Finally, the consequence relation $\models_{NL}$ between an NL theory Γ_L (Law statements) and a logical combination of narrative consequence (Nec and Pos) and verification (Within) statements α is as follows. Γ_B denotes the base theory.

$$
\Gamma_L \models_{NL} \alpha \text{ iff } \Gamma_B \wedge \forall \delta, m[\![\Gamma_L]\!]^{\delta}_{m} \models \forall \delta [\![\alpha]\!]^{\delta}_{\epsilon} \tag{39}
$$

Example: If Γ_L is taken to be the laws (2–3) and α is taken to be the narrative consequence statement (4), then a consequence proof can be made as follows, using natural deduction. Start with Γ_B and the translation of

Γ_L, that is to say (36), as premises. Next, show the translation of (4), that is to say (33), which is a universally quantified conditional, by assuming

$$Eq(d_1, d_2, \epsilon) \wedge Wb(d_2) \wedge De(@0, \mathrm{s4}, \mathbf{n}, d_2)$$

for arbitrary developments d_1, d_2. Now, from $De(@0, \mathrm{s4}, \mathbf{n}, d_2)$ one can infer the right-hand-side of (23) instantiated with d_2, which states that the only action occurrences in the given time segment are

$$D(d_2^d, V(d_2^v, \mathrm{s1}), V(d_2^v, \mathrm{s2}), \textsc{load})$$

and

$$D(d_2^d, V(d_2^v, \mathrm{s3}), V(d_2^v, \mathrm{s4}), \textsc{fire}),$$

which occur in that order. From the above, $Wb(d_2)$, the definition of Wb (12) and the law statements (36) can then be inferred $H(d_2^h, V(d_2^v, \mathrm{s2}), \textsc{loaded})$ as an effect of

$$D(d_2^d, V(d_2^v, \mathrm{s1}), V(d_2^v, \mathrm{s2}), \textsc{load}).$$

Further, with the aid of (38), one can infer

$$\forall t[V(d_2^v, \mathrm{s2}) < t \leq V(d_2^v, \mathrm{s3}) \Rightarrow \neg X(d_2^o, t, \textsc{loaded})]$$

and, with the aid of (12(d)), $H(d_2^h, V(d_2^v, \mathrm{s3}), \textsc{loaded})$. From the latter one can in addition infer $\neg H(d_2^h, V(d_2^v, \mathrm{s4}), \textsc{alive})$ as an effect of

$$D(d_2^d, V(d_2^v, \mathrm{s3}), V(d_2^v, \mathrm{s4}), \textsc{fire}).$$

As we have now shown the conditional for arbitrary d_1, d_2, we can infer the universal (33).

Note that the proof did not involve any reference to the second-order axioms (8)–(11). They do come in, however, in proofs for non-consequence, where they can be used to construct developments that serve as counter-examples.

4 Properties and applications

Besides supporting hypothetical reasoning, the fact that narratives are objects in NL permits us to state properties of, and relations between, narratives, and in addition defining different kinds of operations.

Theorem 3 *Let Γ_L be an NL theory, and let n_1, n_2, n_3 be narrative variables (universally quantified). Then the conditions* Within(s, s', n_1), Within(s', s'', n_2) *and* Within(s'', s''', n_3) *imply the following equivalences.*

$$\mathsf{Nec}(\ s, s', n_1, \mathsf{Nec}(\ s', s'', n_2, \alpha\ )\ ) \equiv \mathsf{Nec}(\ s, s'', \mathrm{J}[n_1, n_2, \mathrm{T}], \alpha\ ) \tag{40}$$

$$\mathsf{Pos}(\ s, s', n_1, \mathsf{Pos}(\ s', s'', n_2, \alpha\ )\ ) \equiv \mathsf{Pos}(\ s, s'', \mathrm{J}[n_1, n_2, \mathrm{T}], \alpha\ ) \tag{41}$$

$$\mathsf{Nec}(\ s, s', n_1, \mathsf{Nec}(\ s'', s''', n_3, \alpha\ )\ ) \equiv \mathsf{Nec}(\ s'', s''', n_3, \mathsf{Nec}(\ s, s', n_1, \alpha\ )\ ) \tag{42}$$

$$\mathsf{Pos}(\ s, s', n_1, \mathsf{Pos}(\ s'', s''', n_3, \alpha\ )\ ) \equiv \mathsf{Pos}(\ s'', s''', n_3, \mathsf{Pos}(\ s, s', n_1, \alpha\ )\ ) \tag{43}$$

$$\mathsf{Nec}(\ s, s', n_1, \alpha\ ) \wedge \mathsf{Nec}(\ s, s', n_1, \alpha'\ ) \equiv \mathsf{Nec}(\ s, s', n_1, \alpha \wedge \alpha'\ ) \tag{44}$$

$$\mathsf{Pos}(\ s, s', n_1, \alpha\ ) \vee \mathsf{Pos}(\ s, s', n_1, \alpha'\ ) \equiv \mathsf{Pos}(\ s, s', n_1, \alpha \vee \alpha'\ ) \tag{45}$$

Proofs. Each of the equivalences can be proved by translating them to PC, and proving them using Γ_B, the translation of Γ_L and the translations of the Within-conditions as premises, and δ and m as the current development and narrative terms. α^T_δ denotes the translation of α.

Equivalence (40) is translated to

$$\begin{array}{l}\forall\delta'[(Eq(\delta,\delta',m) \wedge Wb(\delta') \wedge De(s,s',n_1,\delta')) \Rightarrow \\ \quad \forall\delta''[(Eq(\delta',\delta'',\mathrm{E}(m,s,s',n_1)) \wedge Wb(\delta'') \wedge De(s',s'',n_2,\delta'')) \Rightarrow \\ \quad \alpha^T_{\delta''}]] \equiv \\ \forall\delta'[(Eq(\delta,\delta',m) \wedge Wb(\delta') \wedge De(s,s'',\mathrm{J}[n_1,n_2,\mathrm{T}],\delta')) \Rightarrow \alpha^T_{\delta'}].\end{array}$$

From left to right: 1. Assume the left-hand-side of the equivalence. 1.1. Assume that $(Eq(\delta,d_1,m) \wedge Wb(d_1) \wedge De(s,s'',\mathrm{J}[n_1,n_2,\mathrm{T}],d_1))$ holds for some arbitrary d_1. From that assumption and the definition of De and the Within-conditions, it follows that $De(s,s',n_1,d_1)$ and $De(s',s'',n_2,d_1)$. From the definition of Eq, it follows that $Eq(d_1,d_1,\mathrm{E}(m,s,s',n_1))$. Now, both the outer and inner antecedents of assumption 1 are true for d_1, so we can infer $\alpha^T_{d_1}$. Exit subproof 1.1 and generalize to

$$\forall\delta'[(Eq(\delta,\delta',m) \wedge Wb(\delta') \wedge De(s,s',\mathrm{J}[n_1,n_2,\mathrm{T}],\delta')) \Rightarrow \alpha^T_{\delta'}].$$

End of subproof 1.
From right to left: 2. Assume the right-hand-side of the equivalence. 2.1. Assume

$$(Eq(\delta, d_1, m) \wedge Wb(d_1) \wedge De(s, s', n_1, d_1)).$$

2.1.1. Assume

$$(Eq(d_1, d_2, \mathrm{E}(m, s, s', n_1)) \wedge Wb(d_2) \wedge De(s', s'', n_2, d_2)).$$

From $Eq(\delta, d_1, m)$, $Eq(d_1, d_2, \mathrm{E}(m, s, s', n_1))$ and the definition of Eq it follows that $Eq(\delta, d_2, m)$. $Wb(d_2)$ is already given. From $De(s, s', n_1, d_1)$, the definition of Eq and $Eq(d_1, d_2, \mathrm{E}(m, s, s', n_1))$ it follows that $De(s, s', n_1, d_2)$, and from that, $De(s', s'', n_2, d_2)$ and the definition of De it follows that $De(s, s'', \mathrm{J}[n_1, n_2, \mathrm{T}], d_2)$. Now, the antecedent of assumption 2 is true for d_2, so we can infer $\alpha^T_{d_2}$. Exit subproof 2.1.1 and 2.1, and generalize, and we have

$$\begin{array}{l}\forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s, s', n_1, \delta')) \Rightarrow \\ \quad \forall\delta''[(Eq(\delta', \delta'', \mathrm{E}(m, s, s', n_1)) \wedge Wb(\delta'') \wedge De(s', s'', n_2, \delta'')) \Rightarrow \\ \quad \alpha^T_{\delta''}]].\end{array}$$

End of subproof 2. Consequently, the equivalence holds.
Equivalence (42) is translated to

$$\begin{array}{l}\forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s, s', n_1, \delta')) \Rightarrow \\ \quad \forall\delta''[(Eq(\delta', \delta'', \mathrm{E}(m, s, s', n_1)) \wedge Wb(\delta'') \wedge De(s'', s''', n_3, \delta'')) \Rightarrow \\ \quad \alpha^T_{\delta''}]] \equiv \\ \forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s'', s''', n_3, \delta')) \Rightarrow \\ \quad \forall\delta''[(Eq(\delta', \delta'', \mathrm{E}(m, s, s', n_3)) \wedge Wb(\delta'') \wedge De(s, s'', n_1, \delta'')) \Rightarrow \\ \quad \alpha^T_{\delta''}]].\end{array}$$

From left to right:
1. Assume the left-hand-side of the equivalence. 1.1. Assume

$$(Eq(\delta, d_1, m) \wedge Wb(d_1) \wedge De(s'', s''', n_3, d_1)).$$

1.1.1. Assume

$$(Eq(d_1, d_2, \mathrm{E}(m, s'', s''', n_3)) \wedge Wb(d_2) \wedge De(s, s', n_1, d_2)).$$

From the definition of Eq, 1.1. and 1.1.1. it follows that $Eq(\delta, d_2, m)$. $Wb(d_2)$ and $De(s, s', n_1, d_2)$ are already in 1.1.1. $Eq(d_2, d_2, \mathrm{E}(m, s, s', n_1))$ follows from the definition of Eq. From $Eq(d_1, d_2, \mathrm{E}(m, s'', s''', n_3))$ and

$De(s'', s''', n_3, d_1)$ it follows that $De(s'', s''', n_3, d_2)$. Thus, both the outer and the inner antecedents in assumption 1 are true for d_2. Therefore, we can derive $\alpha^T_{d_2}$. Exit subproofs 1.1.1. and 1.1. and generalize, and we have

$$\forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s'', s''', n_3, \delta')) \Rightarrow \\ \forall\delta''[(Eq(\delta', \delta'', \mathrm{E}(m, s, s', n_3)) \wedge Wb(\delta'') \wedge De(s, s'', n_1, \delta'')) \Rightarrow \\ \alpha^T_{\delta''}]].$$

End of subproof 1. From right to left is exactly the same; just switch the narrative and temporal terms.

Equivalence (44) is translated to

$$(\forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s, s', n_1, \delta')) \Rightarrow \alpha^T_{\delta'}] \wedge \\ \forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s, s', n_1, \delta')) \Rightarrow (\alpha')^T_{\delta'}]) \equiv \\ \forall\delta'[(Eq(\delta, \delta', m) \wedge Wb(\delta') \wedge De(s, s', n_1, \delta')) \Rightarrow (\alpha^T_{\delta'} \wedge (\alpha')^T_{\delta'})]$$

and should be obvious.
The remaining equivalences (41), (43), (45) follow immediately from (40), (42), (44) respectively and the definition of $\mathsf{Pos}(\ s, s', n,\ \alpha\ )$ as $\neg\mathsf{Nec}(\ s, s', n,\ \neg\alpha\ )$.

An important potential application of a formalism that supports reasoning about alternative choices of actions is to formalize higher-level reasoning tasks such as explanation and planning. Both the concept of an explanation and the concept of a plan can be readily formalized in NL, as illustrated by the following definitions. Explanation is the task, given some observation α at an initial time $\mathbf{s}_1$ and some observation β at a later time $\mathbf{s}_2$, to find a narrative n that explains how β came about.

$$\mathsf{Expl}(\alpha, \beta, \mathbf{s}_1, \mathbf{s}_2, n) =_{def} \\ \mathsf{Pos}(\ \mathbf{s}_1, \mathbf{s}_2, n,\ \alpha \wedge \beta\ ) \wedge \mathsf{Within}(\ \mathbf{s}_1, \mathbf{s}_2, n\ ) \tag{46}$$

For instance, the death of the turkey can be explained as follows.

$$\mathsf{Expl}(\mathrm{H}(@0, \mathrm{ALIVE}),\ \neg\mathrm{H}(\mathrm{S3}, \mathrm{ALIVE}),\ @0,\ \mathrm{S3}, \\ \mathrm{J}[\mathrm{D}(\mathrm{S1}, \mathrm{S2}, \mathrm{FIRE}), (@0 < \mathrm{S1}) \wedge (\mathrm{S2} \leq \mathrm{S3})]) \tag{47}$$

However, a resurrection of the turkey would be inexplicable.

$$\neg\exists n[\mathsf{Expl}(\neg\mathrm{H}(\mathrm{S0}, \mathrm{ALIVE}),\ \mathrm{H}(\mathrm{S3}, \mathrm{ALIVE}),\ \mathrm{S0},\ \mathrm{S3},\ n)] \tag{48}$$

Similarly, plan synthesis is the task, given some observation α at an initial time $\mathbf{s}_1$ and some goal β at a later time $\mathbf{s}_2$, to find a narrative $\mathbf{n}$ that is guaranteed to achieve the goal.

$$\mathsf{Plan}(\alpha, \beta, \mathbf{s}_1, \mathbf{s}_2, \mathbf{n}) =_{def} \\ \mathsf{Nec}(\ \mathbf{s}_1, \mathbf{s}_2, \mathbf{n},\ \alpha \Rightarrow \beta\ ) \wedge \mathsf{Pos}(\ \mathbf{s}_1, \mathbf{s}_2, \mathbf{n},\ \mathrm{T}\ ) \wedge \\ \mathsf{Within}(\ \mathbf{s}_1, \mathbf{s}_2, \mathbf{n}\ ) \tag{49}$$

For instance, the following is a plan for killing the turkey that will work independently of the initial state.

$$\begin{array}{l}\mathsf{Plan}(\mathrm{T},\ \neg\mathrm{H}(\mathrm{S4},\ \textsc{alive}),\ \mathrm{S0},\ \mathrm{S4},\\ \quad \mathrm{J}[\mathrm{D}(\mathrm{S1},\mathrm{S2},\textsc{load}),\mathrm{D}(\mathrm{S3},\mathrm{S4},\textsc{fire}),\\ \quad\ (\mathrm{S0}<\mathrm{S1})\wedge(\mathrm{S2}\leq\mathrm{S3})])\end{array} \tag{50}$$

More elaborate definitions involving, for instance, preferences between explanations or plans and constraints on the contents of explanations and plans are also possible. Further, the fact that plans are composite objects in NL facilitates the description of operations such as merging or modifying plans.

It has been argued [120] that deduction-based plan synthesis, i.e. plan synthesis as a deductive existence proof (like in SC), does not require any consistency checks. It should therefore be preferable to abduction-based plan synthesis, i.e. plan synthesis as finding a sentence that entails the goal (like in most narrative-based approaches) and that has to be checked for consistency. Yet, notice that the definition of a plan above involves a narrative consistency condition $\mathsf{Pos}(\ s_1, s_2, n,\ \mathrm{T}\ )$. Narratives in NL are richer objects than action sequences in SC and can involve ordering constraints, and it is possible to construct narratives (e.g. $\mathrm{J}[(\mathrm{S1}<\mathrm{S2})\wedge(\mathrm{S2}<\mathrm{S1})]$) that do not describe any possible development of the world. The same relation holds also for abduction-based approaches (and the operator-based approaches that are common in the planning literature): totally ordered plans (sequences as in SC) are consistent by virtue of their form (provided the actions involved are consistent) whereas partially order plans can contain non-satisfiable ordering constraints and therefore have to be checked for (logical) consistency. Therefore, it is doubtful whether anything is gained in terms of efficiency by formulating plan synthesis deductively instead of abductively.

Nevertheless, a formalism like NL can be of great value for representing plan synthesis and other reasoning tasks. Further, it can be valuable as a compact representation of procedural information, that is information of what actions to execute to solve specific tasks. For instance, (50) provides a procedure for killing turkeys. With such knowledge explicitly represented, an agent does not have to construct a plan from scratch each time it wants to kill a turkey, and in particular it does not have to infer details of the plan such as what holds at what time-point. Further, by extending NL narratives with operations for, for instance, tests and conditional choices, NL could serve as a language for formalizing task execution languages such as Firby's RAPS [40], and even for reasoning about tasks (e.g. planning) in such a language. How to do this is an interesting direction for future work.

5 Conclusions

In order to provide a suitable foundation for theories about such things as plans and explanations in dynamic environments, there are a number of features that are desirable in a formalism for action and change. First, there is often the need for metric time, and for expressing nontrivial temporal properties and relations. Second, it is desirable to have support for reasoning about alternative choices of actions, and to consider such choices as objects in the logic. Third, the alternative ways the world can develop relative to a specific choice of actions should also be represented on the object level. As we stated in the introduction, these three features have been integrated in NL, and it is the introduction of narratives as objects that makes this integration possible. Narratives are entities that are separate from the underlying metric time structure, and this makes it possible to encode rich temporal relations. Due to the narrative-time separation, it should in principle also be possible to change the underlying temporal structure (e.g. to intervals) without any major modifications. Further, the fact that narratives are objects in NL makes it possible to actually reason *about* narratives as opposed to reasoning *within* a specific narrative which is the case in most other narrative-based formalisms. The encoding of the Nec and Pos operators in terms of quantification over developments makes it possible to reason about necessary and possible consequences of a narrative.

It has previously been shown that event-based branching time is not always sufficient for reasoning about alternative choices of actions [62]. In this paper, we have shown that it is not necessary for that purpose either. Actually, we claim that the temporal structure is of secondary importance for the ability to reason about alternative action choices.

NL is defined as a macro language, which is then mapped down to a PC theory and complemented with a number of PC axioms (some of them second-order). Its basis in standard logic is one of many features that distinguishes NL from the work of Pelavin [109], which is the only work with somewhat similar intentions that we are aware of. Other related work, besides what is mentioned in the introduction, includes approaches to combining linear and branching time, such as work by McDermott [98], Sandewall [124], and efforts to bridge the gap between existing narrative-based and branching event-based formalisms [102, 71, 14]. Finally, the solution to the frame problem in NL is an implementation of the PMON entailment policy [124, 26]. PMON has formally been shown to be applicable to worlds with non-overlapping actions with duration and context-dependent and nondeterministic effects, and has recently been extended to deal with ramification

[48], qualification [30] and concurrency [64].

Acknowledgments

This work has been supported by grants from the Wallenberg Foundation and the Swedish Council for Engineering Sciences (TFR). Many thanks to Andrzej Szalas, Patrick Doherty, Silvia Coradeschi, Patrik Haslum and the anonymous referees for comments and advice.

Appendix A

Foundational axioms of TAL

The following is an account for the foundational axioms Γ_{fnd} of TAL.

Axioms for value sorts. For theories with finite value domains, the following unique names axioms hold for each value sort $\mathcal{V}_i = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$:

$$\bigwedge_{1 \leq k < l \leq n} \mathsf{v}_k \neq \mathsf{v}_l \tag{1}$$

In addition, the following domain closure axiom holds for each finite value sort $\mathcal{V}_i$.

$$\forall v \bigvee_{1 \leq l \leq n} v = \mathsf{v}_l \tag{2}$$

Axioms for fluent sorts. For domains with a finite number of fluent sorts $\mathcal{F}_1, \ldots, \mathcal{F}_n$ the following unique names axioms hold:

$$\bigwedge_{1 \leq k < l \leq n} \forall \overline{v}_k, \overline{w}_l [\mathsf{f}_k(\overline{v}_k) \neq \mathsf{f}_l(\overline{w}_l)] \tag{3}$$

$$\bigwedge_{1 \leq l \leq n} \forall \overline{v}_l, \overline{v}'_l [\mathsf{f}_l(\overline{v}_l) = \mathsf{f}_l(\overline{v}'_l) \Rightarrow \bigwedge_i v_{li} = v'_{li}] \tag{4}$$

In addition, the following domain closure axiom holds for each fluent sort $\mathcal{F}_i$.

$$\forall f_i \bigvee_{1 \leq l \leq n} \exists \overline{v}_l [f = \mathsf{f}_l(\overline{v}_l)] \tag{5}$$

Axioms for the action sort $\mathcal{A}$. For domains with a finite number of action types $\mathsf{A}_1, \ldots, \mathsf{A}_n$ the following unique names axiom holds:

$$\bigwedge_{1 \leq k < l \leq n} \forall \overline{v}_k, \overline{w}_l [\mathsf{A}_k(\overline{v}_k) \neq \mathsf{A}_l(\overline{w}_l)] \tag{6}$$

$$\bigwedge_{1\leq l\leq n} \forall \overline{v}_l, \overline{v}'_l[\mathsf{A}_l(\overline{v}_l) = \mathsf{A}_l(\overline{v}'_l) \Rightarrow \bigwedge_i v_{ki} = v'_{ki}] \tag{7}$$

In addition, the following domain closure axiom holds for the action sort:

$$\forall a \bigvee_{1\leq l\leq n} \exists \overline{v}_l[a = \mathsf{a}_l(\overline{v}_l)] \tag{8}$$

Axioms for the temporal domain. A number of different axiomatizations for the temporal domain can be used. Currently we use the axioms for Peano arithmetics without multiplication. When reductions of circumscriptive theories to first-order theories is the focus of research, we sometimes replace the induction axiom with an induction schema.

Bibliography

[1] *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996. AAAI Press, Menlo Park, California.

[2] *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island, 1997. AAAI Press, Menlo Park, California.

[3] Allen, Kautz, Pelavin, and Tenenberg, editors. *Reasoning About Plans.* Morgan Kaufmann, 1991.

[4] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning.* Morgan Kaufmann, 1990.

[5] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984. Reprinted in [4].

[6] James F. Allen. Temporal reasoning and planning. In Allen et al. [3].

[7] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, 1992.

[8] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 141–153, 1996.

[9] Andrew B. Baker. A simple solution to the Yale shooting problem. In KR89 [73].

[10] C. Baral, A. Gabaldon, and A. Provetti. Formalizing narratives using nested circumscription. In AAAI96 [1].

[11] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, 1993. Morgan Kaufmann.

[12] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[13] C. Baral, M. Gelfond, and A. Provetti. Representing actions: Laws, observation and hypothesis. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[14] Kristof van Belleghem, Marc Denecker, and Danny de Schrege. On the relation between situation calculus and event calculus. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[15] Susanne Biundo. Present-day deductive planning. In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning*. IOS Press, Amsterdam, 1995.

[16] Marcus Bjäreland and Lars Karlsson. Reasoning by regression: Pre- and postdiction procedures for logics of action and change with nondeterminism. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, August, 1997. AAAI press.

[17] Sven-Erik Bornscheuer and Michael Thielscher. Representing Concurrent Actions and Solving Conflicts. In B. Nebel and L. Dreschler-Fischer, editors, *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 861 of *LNAI*, pages 16–27, Saarbrücken, Germany, September 1994. Springer.

[18] Sven-Erik Bornscheuer and Michael Thielscher. Explicit and implicit indeterminism. Reasoning about uncertain and contradictory specifications of dynamic systems. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[19] Xiao Jun Chen and Giuseppe De Giacomo. Reasoning about nondeterministic and concurrent actions: a process algebra approach. *Artificial Intelligence*, 107:63–98, 1999.

[20] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[21] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[22] Ernest Davis. Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5):721–766, October 1994.

[23] R. Davis, H. Shrobe, and P. Szolovitz. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[24] G. De Giacomo, Y. Lesperance, and H.J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1221–1226, Nagoya, 1997. Morgan Kaufmann.

[25] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[26] Patrick Doherty. Reasoning about action and change using occlusion. In *Proceedings of the Eleventh European Conference on Artificial Intelligence*, Amsterdam, 1994. John Wiley & Sons.

[27] Patrick Doherty. PMON+: A fluent logic for action and change. Technical Report 96-33, Department of Computer and Information Science, Linköping University, 1996. Available at Linköping University Electronic Press, `http://www.ep.liu.se/`.

[28] Patrick Doherty and Joakim Gustafsson. Delayed effects of actions = direct effects + causal rules. Linköping University Electronic Press. Available at `http://www.ep.liu.se/`., 1998.

[29] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL: Temporal action logics language. Specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 1999.

[30] Patrick Doherty and Jonas Kvarnström. Tackling the qualification problem using fluent dependency constraints: Preliminary report. In *TIME'98*, Florida, 1998.

[31] Patrick Doherty and Jonas Kvarnström. TALplanner: a narrative temporal logic-based forward-chaining planner. In *TIME'99*, 1999.

[32] Patrick Doherty and Witold Lukaszewicz. Circumscribing features and fluents. In *Proceedings of the 1st International Conference on Temporal Reasoning*, pages 82–100. Springer, 1994.

[33] Patrick Doherty and Witold Lukaszewicz. Explaining explanation closure. In *ISMIS'96: Foundations of Intelligent Systems*, Lecture Notes for Artificial Intelligence. Springer Verlag, 1996.

[34] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 16, pages 997–1072. MIT Press, 1990.

[35] E.A. Emerson and E.M. Clarke. Characterizing correctness properties in parallel programs as fixpoints. In *Proceedings of the Seventh International Colloquium on Automata, Languages, and Programming*, number 85 in Lecture Notes in Computer Science. Springer Verlag, 1980.

[36] E.A. Emerson and E.M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[37] K. Eshghi. Abductive planning with event calculus. In *Proceedings of the Fifth International Conference on Logic Programming*, pages 562–579, 1988.

[38] Jacques Ferber and Jean-Pierre Müller. Influences and reaction: a model of situated multi-agent systems. In Mario Tokoro, editor, *Proceedings of the Second International Conference on Multi-Agent Systems*, Kyoto, December 1996. AAAI Press.

[39] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[40] James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.

[41] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[42] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24, 1984.

[43] M Gelfond and V Lifschitz. Representing action and change using logic programs. *Journal of Logic Programming*, 17:301–321, 1993.

[44] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In *Working notes, AAAI Spring Symposium, Series. Symposium: Logical Formalization of Commonsense Reasoning*, 1991.

[45] Michael P. Georgeff. Actions, processes, and causality. In Georgeff and Lansky [46].

[46] M.P. Georgeff and A.L. Lansky, editors. *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, 1987.

[47] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: a possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[48] Joakim Gustafsson and Patrick Doherty. Embracing occlusion in specifying the indirect effects of actions. In KR96 [74].

[49] A.R. Haas. The case for domain-specific frame axioms. In *The frame problem in artificial intelligence. Proceedings of the 1987 workshop*, pages 343–348. Morgan Kaufmann, 1987.

[50] Brian Haugh. Simple causal minimization for temporal persistence and projection. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987. AAAI Press, Menlo Park, California.

[51] Gary Hendrix. Modelling simultaneous actions and continuous processes. *Artificial Intelligence*, 4:145–180, 1973.

[52] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Proceedings of the Seventh International Colloquium on Automata, Languages, and Programming*, number 85 in Lecture Notes in Computer Science. Springer Verlag, 1980.

[53] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[54] J.R. Hobbs and R.C. Moore, editors. *Formal Theories of the Commonsense World.* Ablex, Norwood, New Jersey, 1985.

[55] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[56] *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, 1989. Morgan Kaufmann.

[57] *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

[58] Antonias Kakas and Rob Miller. A simple declarative language for describing narratives with actions. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[59] Antonis Kakas and Rob Miller. Reasoning about actions, narratives and ramification. *Electronic Transactions on Artificial Intelligence*, 1:39–72, 1997.

[60] Lars Karlsson. Causal links planning and the systematic approach to action and change. In *Proceedings of the AAAI 96 Workshop on Reasoning about actions, planning and control: bridging the gap*, Portland, Oregon, 1996. AAAI Press.

[61] Lars Karlsson. Planning, truth criteria and the systematic approach to action and change. In *ISMIS'96: Foundations of Intelligent Systems*, Lecture Notes for Artificial Intelligence. Springer Verlag, 1996.

[62] Lars Karlsson. Reasoning with incomplete initial information and nondeterminism in situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, August, 1997. AAAI press.

[63] Lars Karlsson. Anything can happen: on narratives and hypothetical reasoning. In KR98 [75].

[64] Lars Karlsson and Joakim Gustafsson. Reasoning about action in a multi-agent environment. Available at Linköping University Electronic Press, `http://www.ep.liu.se/`, 1997.

[65] Lars Karlsson and Joakim Gustafsson. Reasoning about concurrent interaction. Accepted for publication in *Journal of Logic and Computation*, 1999.

[66] Lars Karlsson, Joakim Gustafsson, and Patrick Doherty. Delayed effects of actions. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, Brighton, 1998. John Wiley & Sons.

[67] Henry A. Kautz. A formal theory of plan recognition and its implementation. In Allen et al. [3].

[68] M. Koubarakis. Complexity results for first-order theories of temporal constraints. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference*, Bonn, 1994. Morgan Kaufmann.

[69] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[70] R.A. Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[71] Robert Kowalski and Fariba Sadra. Reconciliating the event calculus with the situation calculus. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[72] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, pages 333–354, December 1983.

[73] *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference*, Toronto, 1989. Morgan Kaufmann.

[74] *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference*, Cambridge, Massachusetts, 1996. Morgan Kaufmann.

[75] *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference*, Trento, Italy, 1998. Morgan Kaufmann.

[76] Benjamin Kuipers. *Qualitative Reasoning: modeling and simulation with incomplete information.* MIT Press, 1994.

[77] Amy L. Lansky. A representation of parallel activity based on events, structure, and causality. In Georgeff and Lansky [46].

[78] Yves Lesperance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard B. Scherl. A logical approach to high-level robot programming — a progress report. In B. Kuipers, editor, *The AAAI Fall Symposium on Control of the Physical World by Intelligent Systems*, 1994.

[79] Hector J. Levesque. What is planning in the presence of sensing? In AAAI96 [1].

[80] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming, Special Issue: Reasoning about action and change*, 31(1–3), 1997.

[81] Renwei Li and Luís Moniz Pereira. Temporal reasoning and abductive logic programming. In *Proceedings of the Twelfth European Conference on Artificial Intelligence*, Budapest, 1996. John Wiley & Sons.

[82] Vladimir Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, Los Angeles, 1985. Morgan Kaufmann.

[83] Vladimir Lifschitz. Pointwise circumscription. In M. Ginsberg, editor, *Readings in Non-monotonic Reasoning.* Morgan-Kauffmann, 1988.

[84] Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Artificial Intelligence and Logic Programming*, pages 179–193. Oxford University Press, 1993.

[85] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

[86] Vladimir Lifschitz and Arkady Rabinov. Things that change by themselves. In IJCAI89 [56], pages 864–867.

[87] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In IJCAI95 [57].

[88] Fangzhen Lin. Embracing causality in specifying the indeterminate effects of actions. In AAAI96 [1].

[89] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes, 4(5)*, 1994.

[90] Fangzhen Lin and Yoav Shoham. Provably correct theories of actions. *Journal of ACM*, 42(2):293–320, 1995.

[91] Witold Lukaszewicz and Ewa Madalinska-Bugaj. Reasoning about action and change using Dijkstra's semantics for programming languages: Preliminary report. In IJCAI95 [57].

[92] Zohar Manna and Richard Waldinger. A theory of plans. In Georgeff and Lansky [46].

[93] Norman McCain and Hudson Turner. Causal theories of action and change. In AAAI97 [2].

[94] John McCarthy. First order theories of individual concepts and propositions. *Machine Intelligence*, 9, 1979.

[95] John McCarthy. Elaboration tolerance. In *Common Sense 98*, Queen Mary and Westfield College, London, 1998.

[96] John McCarthy and Tom Costello. Combining narratives. In KR98 [75].

[97] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[98] Drew McDermott. Reasoning about plans. In Hobbs and Moore [54].

[99] Sheila McIlraith. Representing actions and state constraints in model-based diagnosis. In AAAI97 [2].

[100] John-Jules Meyer and Patrick Doherty. Preferential action semantics. In *Proceedings of the 1st MODELAGE Workshop*, Lecture Notes in Artificial Intelligence. Springer, 1998.

[101] Rob Miller. Notes on deductive and abductive planning in the event calculus. Unpublished manuscript. Available on WWW: `http://www.doc.ic.ac.uk/~ rsm/`, July 1996.

[102] Rob Miller and Murray Shanahan. Narratives in situation calculus. *Journal of Logic and Computation*, 4(5), October 1994.

[103] Rob Miller and Murray Shanahan. Reasoning about discontinuities in the event calculus. In KR96 [74].

[104] R. Milner. A calculus of communicating systems. In *Lecture Notes in Computer Science*, volume 92. Springer Verlag, 1980.

[105] Robert C. Moore. A formal theory of knowledge and action. In Hobbs and Moore [54], chapter 9.

[106] Leora Morgenstern. The problem with solutions to the frame problem. In K. Ford and Z. Pylyshyn, editors, *The Robot's Dilemma Revisited*. Ablex, 1996.

[107] Leora Morgenstern and Lynn Andrea Stein. Why things go wrong: A formal theory of causal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, 1988. AAAI Press, Menlo Park, California.

[108] Edwin P.D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4:356–372, 1988.

[109] Richard N. Pelavin. Planning with simultaneous actions and external events. In Allen et al. [3].

[110] M. Peot and D. Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*. Morgan Kaufmann, 1992.

[111] Javier Pinto and Raymond Reiter. Temporal reasoning in logic programming: a case for the situation calculus. In David S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 203–221, Budapest, 1993. MIT Press.

[112] Javier A. Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, University of Toronto, 1994.

[113] Javier A. Pinto. Concurrent actions and interactions. In KR98 [75].

[114] Javier A. Pinto. Occurrences and narratives as constraints in the branching structure of the situation calculus. *Journal of Logic and Computation*, 8(6):777–808, 1999.

[115] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[116] V. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science*, 1976.

[117] Alessandro Provetti. Hypothetical reasoning: from situation calculus to event calculus. *Computational Intelligence*, 12(3), 1996.

[118] Ray Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–360. Academic Press, 1991.

[119] Raymond Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, 1993.

[120] Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In KR96 [74].

[121] Raymond Reiter. Sequential, temporal golog. In KR98 [75].

[122] Stanley J. Rosenschein. Plan synthesis: A logical perspective. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 331–37, Vancouver, 1981. Morgan Kaufmann.

[123] Erik Sandewall. Combining logic and differential equations for describing real-world systems. In KR89 [73].

[124] Erik Sandewall. *Features and Fluents.* Oxford Press, 1994.

[125] Erik Sandewall. Logic based modelling of goal-directed behavior. In KR98 [75].

[126] Lenhart Schubert. Monotonic solution of the frame problem in situation calculus: an efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.

[127] Murray Shanahan. Prediction is deduction but explanation is abduction. In IJCAI89 [56], pages 1055–1060.

[128] Murray Shanahan. Representing continuous change in the event calculus. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, Stockholm, 1990. John Wiley & Sons.

[129] Murray Shanahan. A circumscriptive calculus of events. *Artificial Intelligence*, 77:249–284, 1995.

[130] Murray Shanahan. *Solving the Frame Problem.* MIT Press, 1997.

[131] Murray Shanahan. The ramification problem in the event calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, 1999. Morgan Kaufmann.

[132] Yoav Shoham. *Reasoning About Change*. MIT press, 1988.

[133] L.A. Stein and L. Morgenstern. Motivated action theory: Formal theory of causal reasoning. *Artificial Intelligence*, 71(1):1–42, 1994.

[134] E. Ternovskaia. Interval situation calculus. In *Proc. of ECAI'94 Workshop W5 on Logic and Change*, Amsterdam, 1994.

[135] Michael Thielscher. The logic of dynamic systems. In IJCAI95 [57].

[136] Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997.

[137] Michael Thielscher. How (not) to minimize events. In KR98 [75].

[138] Hudson Turner. Representing actions in logic programming: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.

[139] Choong-Ho Yi. Towards assessment of logics for concurrent actions. Presented at the AAAI 1995 Spring Symposium: Extended Theories of Action, Stanford University, 1995.