

Temporal Logics and Temporal Reasoning

Lode Missiaen, 1991

Department of Computer Science, K.U. Leuven

Celestijnenlaan 200A, B-3001 Heverlee

Belgium

Revised by Bern Martens, 1992, 1995, 1996, 1998

Major Revision by Marc Denecker, 2004

Contents

1	Introduction	3
2	Modal Temporal Logic	4
2.1	Syntax of Tense Logic	4
2.2	Proof Theory of Tense Logic	5
2.2.1	Inference rules	5
2.2.2	Axiom schemata	5
2.2.3	Axioms constraining modal operators	6
2.2.4	Summary	8
2.3	Semantics of Tense Logic	8
2.3.1	Possible worlds semantics of TL	8
2.3.2	Transformation of a TL-expression into a FOL-expression	10
2.4	TL Theorem Proving	13
2.4.1	Proof theory (Hilbert style)	13
2.4.2	Model based reasoning	14
2.4.3	FOL techniques	15
2.5	First-order Modal Temporal Logic	15
3	Situation Calculus	18
3.1	State Based Representation	19
3.2	Representation of Situation Calculus in ID-logic	20
3.2.1	Axiomatising the set of situations	20
3.2.2	Frame axioms	22
3.2.3	Formalising the initial situation	25
3.2.4	Action preconditions	26
3.2.5	Summary: TA Situation Calculus	26
3.2.6	The reified solution	26
3.2.7	A block world example	27
3.2.8	Expressiveness of situation calculus	28
3.3	Temporal Reasoning with Situation Calculus	31
3.4	Situation Calculus in Prolog	32
3.5	Prediction, Postdiction	33
3.6	Planning	35
3.7	Difficulties of situation calculus	36
3.7.1	Global states and modularity	37
3.7.2	Total ordering of events	37

1 Introduction

This text presents a selection of existing logical formalisms for reasoning about and representing *time*. Temporal logic and reasoning are needed in Artificial Intelligence applications, in particular *natural language understanding* and *plan generation*, but also in temporal database management systems and software engineering of real-time systems. This course only discusses the use of temporal logic and reasoning for tense markers in natural language (NL) and generation of ordered actions in planning.

This course classifies temporal formalisms as either *modal logic* or *first-order logic* (FOL). In *modal temporal logic*, *modal operators* are used to represent temporal information. A *first-order logical* approach uses a *separate argument* to represent time, either straightforwardly as an extra argument in time-dependent properties, or combined with a *reification* of the latter.

The representation of time can be either *point-based* or *interval-based*, *discrete* or *dense*, *bounded* or *unbounded* in the past and future, *linear* or *branching*.

Apart from the introduction, this text contains four main sections. Section 2 presents a propositional modal temporal logic: *tense logic*. Tense logic represents simple tense markers in natural language. Section 3 presents *Situation Calculus* and Section ?? presents *Event Calculus*. Both are point-based, FOL approaches. We explain both deduction and abduction inference procedures for temporal reasoning. Finally, Section ?? presents an interval-based, FOL approach: *Allen's theory of time*.

References

- [1] D. M. Gabbay, C. J. Hogger and J. A. Robinson (Eds.). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4: Epistemic and Temporal Reasoning*. Oxford Science Publications, 1995

2 Modal Temporal Logic

In *Modal Temporal Logic* (MTL), modal operators designate *temporal connectives* (when, while, before, ...), *temporal adverbs* (now, then, until, ...), and *verb tenses* (past, future, ...). Such modal operators allow for a natural expression of temporal information in natural language.

This section studies a simple propositional Modal Temporal Logic called *Tense Logic* (TL) with only 4 modal operators. We introduce the syntax, the proof theory and the semantics of this language, and explain some theorem proving techniques. The end of this section hints at extending this logic to a predicate Modal Temporal Logic.

2.1 Syntax of Tense Logic

The syntax of a TL-expression, also called a TL-sentence or TL-formula, consists of *proposition calculus* extended with four modal operators: \mathcal{A}_p , \mathcal{A}_f , \mathcal{P} and \mathcal{F} .

$\mathcal{A}_p(\phi)$: ϕ was always true in the past

$\mathcal{A}_f(\phi)$: ϕ is always going to be true in the future

$\mathcal{P}(\phi)$: two possible interpretations;
 ϕ may have been true in the past (subjunctive past), or
 ϕ was true in the past (simple past)

$\mathcal{F}(\phi)$: two possible interpretations;
 ϕ may be true in the future (subjunctive future), or
 ϕ will be true in the future (simple future)

Notice the two possible interpretations of $\mathcal{P}(\phi)$ and $\mathcal{F}(\phi)$: subjunctive or simple. The interpretation chosen will depend upon the constraints put on the modal operators. These constraints are defined by the axioms of the proof theory of the language (Section 2.2).

A TL-sentence states a property with respect to an implicit time point, namely the current time point, that is *now*. For example, the TL-sentence P states that P is true now; the expression $\mathcal{P}(P)$ states that P was true in the past (from now).

A TL-theory consists of a set of TL-sentences.

The modal operators allow to represent the verb tenses of natural language sentences explicitly.

Example 1

- *I will have done it.* (future perfect)
 $\mathcal{F}(\mathcal{P}(\text{do}(\text{I}, \text{it})))$
- *I was going to do it.* (past progressive)
 $\mathcal{P}(\mathcal{F}(\text{do}(\text{I}, \text{it})))$

2.2 Proof Theory of Tense Logic

Tense logic proof theory is composed of two sets: inference rules and axiom schemata.

2.2.1 Inference rules

Inference rules contain conditions and the conclusions which can be drawn when the conditions are met. Both conditions and conclusions are logical sentences, here represented by Greek letters.

(MP) from ϕ and $(\phi \rightarrow \psi)$ infer ψ	(Modus Ponens)
(NEC _p) from <i>axiom</i> (ϕ) infer $\mathcal{A}_p(\phi)$	(Past Necessity)
(NEC _f) from <i>axiom</i> (ϕ) infer $\mathcal{A}_f(\phi)$	(Future Necessity)

2.2.2 Axiom schemata

The axioms of proof theory of a logic are often defined by *axiom schemata*. An axiom schemata is represented as a formula containing formula parameters. For example, an axiom schemata of first order logic is $\phi \wedge (\phi \supset \psi) \supset \psi$ and its formula parameters are ψ and ϕ . An axiom schemata represents a set of *axioms* namely all those that can be obtained by substituting formulas for formula parameters. So, two instances of the axiom schemata are

$$P \wedge (P \supset Q) \supset Q$$

and

$$(P \vee Q) \wedge ((P \vee Q) \supset (P \wedge Q \wedge R)) \supset (P \wedge Q \wedge R)$$

In temporal logic, the axiom schemata of proof theory represent temporal properties which are eternally true, i.e. true in every time point. As we will see later, user defined axioms represent properties that are true in the *current time point*, i.e. which are true *now*.

Every proof theory of Tense logic contains a set of proposition calculus axioms (AX0) and three pairs of intuitive, symmetrical axioms (AX1–AX3) which hold true for every pattern. Further axiom schemata can be added to constrain the interpretation of the modal operators (Section 2.2.3). A proof theory containing only (AX0–AX3) is called *minimal temporal logic*.

(AX0) Every tautology of the proposition calculus renders an axiom schema. e.g., $\neg\neg\phi \leftrightarrow \phi$ from $\neg\neg p \leftrightarrow p$

(AX1 _p) $\mathcal{A}_p(\phi \rightarrow \psi) \rightarrow (\mathcal{A}_p(\phi) \rightarrow \mathcal{A}_p(\psi))$
(AX1 _f) $\mathcal{A}_f(\phi \rightarrow \psi) \rightarrow (\mathcal{A}_f(\phi) \rightarrow \mathcal{A}_f(\psi))$

Axiom schemata (AX1_p) and (AX1_f) correspond to the K-axiom in Modal Logic which allows the modal operator to be distributed into the implication: \mathcal{A}_p and \mathcal{A}_f are *normal* modal operators.

$$\text{(AX2}_p\text{)} \quad \phi \rightarrow \mathcal{A}_f(\mathcal{P}(\phi))$$

$$\text{(AX2}_f\text{)} \quad \phi \rightarrow \mathcal{A}_p(\mathcal{F}(\phi))$$

Axiom schema (AX2_p) states that if something is true now, than at any time in the future it must have been true in the past. (AX2_f) is symmetrical to (AX2_p) with respect to past and future.

$$\text{(AX3}_p\text{)} \quad \mathcal{P}(\mathcal{P}(\phi)) \rightarrow \mathcal{P}(\phi)$$

$$\text{(AX3}_f\text{)} \quad \mathcal{F}(\mathcal{F}(\phi)) \rightarrow \mathcal{F}(\phi)$$

Axiom schema (AX3_p) states that if ϕ has been true in the past, then it continues to have been true. (AX3_f) is symmetrical to (AX3_p) with respect to past and future.

2.2.3 Axioms constraining modal operators

The axioms (AX4)–(AX6) determine the topology of tense logic: linear past, branching future, unbounded past and future, dense time. Tense logic does not distinguish between time points or time intervals. In other words, the explanations of the axioms in terms of time points could be given equally well in terms of intervals.

Linear versus Branching Past/Future The branching axioms (AX4_p) and (AX4_f), added to the axiom schemata, constrain the definition of the modal operators \mathcal{P} and \mathcal{F} , respectively.

$$\text{(AX4}_p\text{)} \quad (\mathcal{P}(\phi) \wedge \mathcal{P}(\psi)) \rightarrow \mathcal{P}(\phi \wedge \psi) \vee \mathcal{P}(\mathcal{P}(\phi) \wedge \psi) \vee \mathcal{P}(\phi \wedge \mathcal{P}(\psi))$$

$$\text{(AX4}_f\text{)} \quad (\mathcal{F}(\phi) \wedge \mathcal{F}(\psi)) \rightarrow \mathcal{F}(\phi \wedge \psi) \vee \mathcal{F}(\mathcal{F}(\phi) \wedge \psi) \vee \mathcal{F}(\phi \wedge \mathcal{F}(\psi))$$

(AX4_p) states that if ϕ and ψ were both true in the past, then either

1. at some past time, ϕ and ψ were both true together
2. or, at some past time ψ was true and ϕ had previously been true
3. or, at some past time ϕ was true and ψ had previously been true

(AX4_p) defines time to be linear in the past allowing only one possible past. Employing (AX4_p) defines $\mathcal{P}(\phi)$ as ϕ *was true*. Similarly, employing (AX4_f) would define time to be linear in the future.

In Tense Logic, we want time to branch forward, with multiple possible futures, but we accept the past as linear. Therefore, we reject (AX4_f) and accept (AX4_p), and interpret $\mathcal{F}(\phi)$ as ϕ *may be true*.

Exercise 2 *The admission of (AX_{4p}) and the rejection of (AX_{4f}) stems from one way of thinking about the present: everything that has happened is fixed, but the future is open. Show that this assumption is not always appropriate and that presence of (AX_{4p}) and (AX_{4f}) is arguable and depends on its usefulness for any particular application.*

Unbounded Past/Future

$$(AX_{5p}) \mathcal{A}_p(\phi) \rightarrow \mathcal{P}(\phi)$$

$$(AX_{5f}) \mathcal{A}_f(\phi) \rightarrow \mathcal{F}(\phi)$$

(AX_{5p}) states that everything which was true at all times in the past was indeed true at some time in the past. The only way for (AX_{5p}) to be false is when the time is bounded in the past, i.e., if there is a first instant of time. With respect to that first time point, everything is trivially true at all times before that time point (because there are no such time points) but there is no past time at which it is true. As a result, (AX_{5p}) implies that time is unbounded in the past.

Similarly, (AX_{5f}) implies that time is unbounded in the future, i.e., that there is no last time point.

Denseness of Time

$$(AX_{6p}) \mathcal{P}(\phi) \rightarrow \mathcal{P}(\mathcal{P}(\phi))$$

$$(AX_{6f}) \mathcal{F}(\phi) \rightarrow \mathcal{F}(\mathcal{F}(\phi))$$

Axiom (AX_{6p}) states that if ϕ is true in the past then there is a time point between now and then at which ϕ is still true in the past and similarly for sentences that are true in the future, (AX_{6f}) . Together, (AX_6) states that for any two time points that are ordered in time, there is a third time point between these two time points, or in other words, that a time line is dense.

Summary: Topology of Time The topology of time is determined by the following characteristics of time:

- *point-based* versus *interval-based*
- *bounded* versus *unbounded*
- *linear* versus *branching*
- *discrete* versus *dense*

Although temporal logics do not make a choice between point-based and interval-based topology, there is no specific support for interval-based topology. So, we are allowed to think about time points as (small) intervals in which no changes occur to the world. However, there is no way in which we can express typical interval-based properties, such as that two intervals overlap, or that some properties stay

true in an interval, etc. In practice, this means that it is easier to think of these temporal logics as point-based. Also formal semantics are point-based: time corresponds to points, not to intervals. Other theories of time such as situation and event calculus are point-based as well. Allens theory of time (Section ??) is interval-based.

2.2.4 Summary

In Tense logic, the accepted axiom schemata are AX0-AX3, AX4_p, AX5, AX6. It is a logic with linear past, branching future. Its time topology is point-based, with unbounded past and future and dense time.

2.3 Semantics of Tense Logic

We present two semantics for Tense Logic. One is a standard a possible world semantics of modal logic. The second one is given by a transformation into FOL.

2.3.1 Possible worlds semantics of TL

We can define a *Possible Worlds Semantics* for TL. A possible world structure, called a time frame, specifies a set of time points, a precedence relation \prec , a current time point t_0 and for each time point t a set of propositions that are true at time t .

This semantics is an instance of the more general principle of *Possible Worlds Semantics* of Modal Logic. A possible world structure of modal logic consists of a set of worlds, an accessibility relation between them and for each world, a set of propositions. In the semantics of Tense logic, the worlds correspond to the time points, and the accessibility relation between the possible worlds correspond to the precedence relation on time points, i.e., \prec .

We define a *temporal frame* $M = (\mathcal{T}, \prec, t_0, \mathcal{I})$ of a set of propositional symbols τ , as follows:

- \mathcal{T} : a set of time points
- \prec : a precedence relation on the time points
- $t_0 \in \mathcal{T}$: the current time point (now!)
- $\mathcal{I} : \mathcal{T} \rightarrow 2^\tau$: a mapping from time points t to sets of propositions true at t

$I(t)$ represents a *state*: the set of properties that hold true simultaneously.

We define when temporal frame M satisfies a temporal sentence ϕ or equivalently, when ϕ is true in M , denoted $M \models \phi$. The definition is by induction on the length of the sentence:

- $M \models \phi$ if ϕ is a propositional atom and $\phi \in I(t_0)$
- $M \models \mathcal{P}(\phi)$ if $M[t_0/t_1] \models \phi$, for some $t_1 \prec t_0$ in \mathcal{T}
- $M \models \mathcal{F}(\phi)$ if $M[t_0/t_1] \models \phi$, for some $t_1 \succ t_0$ in \mathcal{T}

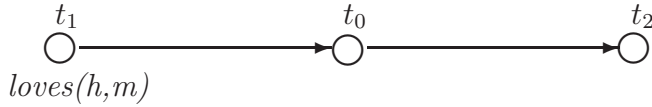
- $M \models \mathcal{A}_p(\phi)$ if $M[t_0/t_1] \models \phi$, for each $t_1 \prec t_0$ in \mathcal{T}
- $M \models \mathcal{A}_f(\phi)$ if $M[t_0/t_1] \models \phi$, for each $t_1 \succ t_0$ in \mathcal{T}
- $M \models \phi \wedge \psi$ if $M \models \phi$ and $M \models \psi$
- $M \models \phi \vee \psi$ if $M \models \phi$ or $M \models \psi$ or both
- $M \models \neg\phi$ if $M \not\models \phi$ and $M \models \psi$

Here $M[t_0/t_1]$ denotes the temporal frame identical to M with exception that the current time point is t_1 rather than t_0 .

Example 3 Consider the NL sentence Harry will have loved Mary.. This can be represented by TL-sentence $\phi = \mathcal{F}(\mathcal{P}(\text{love}(H, M)))$. This sentence is true in $M_1 = (\mathcal{T}_1, \prec_1, t_0, \mathcal{I}_1)$:

- $\mathcal{T}_1 = \{t_0, t_1, t_2\}$
- $\prec_1 = \{t_1 \prec t_0, t_0 \prec t_2, t_1 \prec t_2\}$
- $\mathcal{I}_1(t_0) = \emptyset, \mathcal{I}_1(t_1) = \{\text{loves}(H, M)\}, \mathcal{I}_1(t_2) = \emptyset$

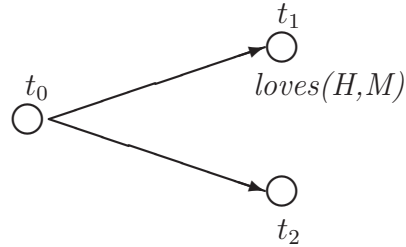
We can represent M_1 as follows:



In M_1 , there exists a time point t_2 later than t_0 for which there exists a time point t_1 that precedes t_2 and at which $\text{loves}(H, M)$ holds.

However, ϕ is false for $M_2 = (\mathcal{T}_2, \prec_2, t_0, \mathcal{I}_2)$:

- $\mathcal{T}_2 = \{t_0, t_1, t_2\}$
- $\prec_2 = \{t_0 \prec t_1, t_0 \prec t_2\}$
- $\mathcal{I}_2(t_0) = \emptyset,$
 $\mathcal{I}_2(t_1) = \{\text{loves}(H, M)\},$
 $\mathcal{I}_2(t_2) = \emptyset$



In M_2 , there is no time point before t_1 or before t_2 at which the proposition $\text{loves}(H, M)$ holds.

Not every temporal frame is acceptable in each temporal logic. Let \mathcal{A} be a subset of the axiom schemata AX0 - AX6, and $\mathcal{L}_{\mathcal{A}}$ the logic defined by the chosen set of axioms. These axioms impose constraints on the temporal frames. Below, we specify these constraints by FOL formulas about \mathcal{T} and \prec .

- (AX0–AX2) impose no constraints on the precedence relation \prec . A *minimal temporal logic* has no constraints on the precedence relation \prec . The axioms (AX0–AX2), together with the inference rules (MP) and (NEC), induce a logic whose theorems hold true at all times in all temporal frames.

- (AX3): Transitivity of \prec

$$\forall t, t_1, t_2 (t \prec t_1 \wedge t_1 \prec t_2 \supset t \prec t_2)$$

Transitivity of the precedence relation on time points is a natural property. AX3 is an uncontroversial axiom schemata.

- (AX4_p): Linear past

$$\forall t, t_1, t_2 (t_1 \prec t \wedge t_2 \prec t \supset t_1 = t_2 \vee t_1 \prec t_2 \vee t_2 \prec t_1)$$

- (AX4_f): Linear future

$$\forall t, t_1, t_2 (t_1 \succ t \wedge t_2 \succ t \supset t_1 = t_2 \vee t_1 \prec t_2 \vee t_2 \prec t_1)$$

- (AX5_p): Unbounded past

$$\forall t \exists t_1 (t_1 \prec t)$$

- (AX5_f): Unbounded future

$$\forall t \exists t_1 (t_1 \succ t)$$

- (AX6): Denseness

$$\forall t, t_1 (t \prec t_1 \supset \exists t_2 (t \prec t_2 \wedge t_2 \prec t_1))$$

Recall that Tense logic satisfies AX0 - AX3, AX4_p, AX5 - AX6. From both (AX5) and (AX6), it follows that every acceptable temporal frame of Tense logic has an infinite number of time points. In the remainder of this section, we will use finite temporal frames for demonstrating certain arguments. **These finite temporal frames must then always be considered as a subset of infinite, dense frames for which the same argument holds!**

The following definitions are simple extensions of the corresponding model semantic definitions in FOL. Let ϕ be a TL-formula. A temporal frame M is a *model for* ϕ if ϕ is true for M . If ϕ has a model, then ϕ is *satisfiable* or *consistent*. Otherwise, it is *unsatisfiable* or *inconsistent*. When every possible temporal frame is a model for ϕ , then ϕ is *valid*. A TL-formula ψ is a *logical consequence* of ϕ (or, ϕ *logically implies* ψ), if every model of ϕ is also a model of ψ . This is written as $\phi \models \psi$. ψ is *logically equivalent* with ϕ if both $\phi \models \psi$ and $\psi \models \phi$ hold.

2.3.2 Transformation of a TL-expression into a FOL-expression

We now define an alternative semantics for Tense logic by providing a translation from TL-sentences into First Order Logical (FOL) expressions.

The transformation of a Tense Logical (TL) expression into a First Order Logical (FOL) expression removes the modal operators from the TL-expression by adding a temporal argument to every proposition. The transformed TL-expression ϕ is written as $\pi(\phi)$.

Given is a TL-vocabulary τ of propositional symbols. The translation maps TL-sentences to FOL formulas in a FOL vocabulary τ' which consists of the following symbols:

- (1) one unary predicate symbol $P(t)$ for each TL-symbol $P \in \tau$
- (2) a binary predicate symbol \prec
- (3) one constant symbol Now
- (4) a set of variables including t_0, t_1, t_2, \dots

With $\phi[t_0/t]$, we denote the formula obtained by replacing every occurrence of t_0 in ϕ by t . This function is only defined if t does not occur in ϕ . The function $\pi(\phi)$ is defined inductively.

- $\pi(\phi) = \phi(t_0)$, **if** ϕ is a propositional atom
- $\pi(\mathcal{P}(\phi)) = (\exists t(t \prec t_0 \wedge \pi(\phi)[t_0/t]))$
- $\pi(\mathcal{F}(\phi)) = (\exists t(t \succ t_0 \wedge \pi(\phi)[t_0/t]))$
- $\pi(\mathcal{A}_p(\phi)) = (\forall t(t \prec t_0 \supset \pi(\phi)[t_0/t]))$
- $\pi(\mathcal{A}_f(\phi)) = (\forall t(t \succ t_0 \supset \pi(\phi)[t_0/t]))$
- $\pi(\phi_1 \wedge \dots \wedge \phi_n) = (\pi(\phi_1) \wedge \dots \wedge \pi(\phi_n))$
- $\pi(\phi_1 \vee \dots \vee \phi_n) = (\pi(\phi_1) \vee \dots \vee \pi(\phi_n))$
- $\pi(\neg\phi) = (\neg\pi(\phi))$
- $\pi(\phi \supset \psi) = (\pi(\phi) \supset \pi(\psi))$
- $\pi(\phi \equiv \psi) = (\pi(\phi) \equiv \pi(\psi))$

The symbol \succ should not be understood as a new predicate symbol but $t \succ t_1$ is used here as a shorthand notation for the atom $t_1 \prec t$. We use this notation to pinpoint symmetries between treatment of past and future.

Example 4 Harry will have loved Mary.

This NL-sentence can be written in TL as $\mathcal{F}(\mathcal{P}(\text{loves}(H, M)))$. We transform this TL-expression into FOL from the inside outwards:

1. $\pi(\text{loves}(H, M)) = \text{loves}(H, M, t_0)$
2. $\pi(\mathcal{P}(\text{loves}(H, M))) = (\exists t_1(t_1 \prec t_0 \wedge \pi(\text{loves}(H, M))[t_0/t_1]))$
 $= (\exists t_1(t_1 \prec t_0 \wedge \text{loves}(H, M, t_1)))$

$$\begin{aligned} 3. \pi(\mathcal{F}(\mathcal{P}(\text{loves}(H, M)))) &= (\exists t_2(t_2 \succ t_0 \wedge \pi(\mathcal{P}(\text{loves}(H, M))[t_0/t_2]))) \\ &= (\exists t_2(t_2 \succ t_0 \wedge (\exists t_1(t_1 \prec t_2 \wedge \text{loves}(H, M, t_1)))) \end{aligned}$$

Notice that the outcome $\pi(\phi)$ contains the free variable t_0 . Therefore, $\pi(\phi)$ is not a meaningful FOL sentence yet. Some additional steps are needed to translate TL-theories in FOL theories.

Let \mathcal{A} be a subset of the axiom schemata AX0 - AX6, and $\mathcal{L}_{\mathcal{A}}$ the logic defined by these axiom schemata. Let T be a set of temporal logic sentences. We define $\pi_{\mathcal{A}}(T)$ as the FOL theory consisting of the following axioms:

- (1) the set of time topology axioms corresponding to the axiom schemata in \mathcal{A}
- (2) the set of formulas $\pi(\phi)[t_o/Now]$, for each $\phi \in T$.

The theory $\pi_{\mathcal{A}}(T)$ has models. It is easy to see the correspondence between its models and the temporal frames satisfying T . With each temporal frame $M = (\mathcal{T}, \prec, t_0, I)$, we can associate a structure $J = \pi(M)$ of the FOL vocabulary τ' :

- (1) the domain $D_J = \mathcal{T}$;
- (2) $\prec^J = \prec$;
- (3) $Now^J = t_0$;
- (4) for each predicate symbol $P \in \tau'$, $P^J = \{t : P \in I(t)\}$.

Or, P^J is the set of all time points at which P is true in M . Vice versa, with each τ' -structure J , a unique temporal frame M corresponds such that $\pi(M) = J$.

It is easy to understand and to show that for arbitrary temporal theory T and temporal frame M , it holds that $M \models_{\mathcal{A}} T$ iff $\pi(M) \models \pi_{\mathcal{A}}(T)$.

Exercise 5 Transform the following NL-sentences into TL and into FOL. Write the FOL-sentences directly and via the function π , and compare your results.

- *Mary walks.*
- *Mary will have walked.*
- *Mary walked.*
- *Mary was going to walk.*
- *Mary had walked.*
- *Whenever she visits me, she starts arguing.*
- *Mary will walk.*

In Section 2.2 we saw that the axioms of the proof theory determined the time topology. We now explain this. It is interesting to apply the transformation π also to the instances of the axiom schemata \mathcal{A} of a temporal logic $\mathcal{L}_{\mathcal{A}}$. Recall that axiom schemata describe properties that hold in *all* time points. Therefore, they should be translated into formulas not about *Now* but true in all time points.

In some cases AX0-AX2, we obtain FOL tautologies. Consider for example (AX2_f): $\phi \rightarrow \mathcal{A}_p(\mathcal{F}(\phi))$. Transformation results in

$$\forall t_0(\pi(\phi) \supset (\forall t_1(t_1 \prec t_0 \supset (\exists t_2(t_2 \succ t_1 \wedge \pi(\phi)[t_0/t_2])))$$

This expression holds for all time points t_0 . Notice it is trivially satisfied (take $t_2 = t_0$) and therefore does not impose any constraint on the precedence relation \prec .

But in the remaining cases AX3-AX6, we obtain sets of formulas which constrain the time topologies. For example,

$$(AX3_p): \mathcal{P}(\mathcal{P}(\phi)) \supset \mathcal{P}(\phi)$$

transforms into

$$\forall t_0(\exists t_1(t_1 \prec t_0 \wedge (\exists t_2(t_2 \prec t_1 \wedge \pi(\phi)[t_0/t_2]))) \supset (\exists t(t \prec t_0 \wedge \pi(\phi)[t_0/t])))$$

Such sets of FOL-axioms (one for each ϕ) actually corresponds to a second order formula:

$$\forall X \forall t_0(\exists t_1(t_1 \prec t_0 \wedge (\exists t_2(t_2 \prec t_1 \wedge X(t_2)))) \supset (\exists t(t \prec t_0 \wedge X(t))))$$

By application of second order theorem proving techniques which are beyond the goals of this tutorial, the set variable X can be removed. This yields the following formula

$$\forall t \forall t_1 \forall t_2((t \prec t_1 \wedge t_1 \prec t_2) \supset t \prec t_2)$$

This formula is nothing else than the *transitivity law*, i.e. the time topology axiom corresponding to AX3! Actually, the same holds for each of the axiom schemata AX3-AX6: by translation via π and removal of the parameter ϕ , we obtain the corresponding time topology axiom.

Exercise 6 (Optional) Transform the axiom schemata (AX1)–(AX6) of Section 2.2 into FOL.

Exercise 7 (Optional) Write a LISP or Prolog program that implements the function π . In what computing system could such an implementation be a component ?

2.4 TL Theorem Proving

Given a consistent theory T of TL formulae, theorem proving shows that a TL formula ϕ is a semantic consequence of T : $T \models \phi$. There exist several approaches to this task: either performing the proofs within TL, or reasoning in terms of temporal frames, or transforming the problem into FOL and applying techniques from FOL theorem proving.

2.4.1 Proof theory (Hilbert style)

Suppose we want to prove $T \models \phi$. A forward chaining proof procedure in TL can be defined as follows. A proof of ϕ is a finite sequence of TL expressions in which

1. ϕ is the last element of the sequence, and

2. every member of the sequence is a member of T , a logical axiom (AX0)–(AX6), or the result of applying (MP), (NEC_p) or (NEC_f) to expressions earlier in the sequence.

By $T \vdash \phi$, we express that there exists a proof of ϕ from T .

The most important basic property about proof theory and semantics of Tense Logic is that the proof theory of Tense Logic is *sound and complete*. This means that for every TL-theory T and TL-formula ϕ , $T \models \phi$ iff $T \vdash \phi$.

Such a proof procedure gives rise to an enormous search space of possible sequences. Automating it would require powerful heuristics to control the search space. In some cases, heuristics render the proof procedure incomplete.

Also for humans, although a given proof of this kind can quite easily be verified, constructing such proofs from scratch is difficult.

2.4.2 Model based reasoning

In Section 2.3.2, we explained a method to transform a TL expression into FOL by introducing a time argument. We also saw how a possible world semantics of TL could be defined. Humans often find it convenient to verify validity, satisfiability, semantical entailment, etc. in TL directly in terms of these possible worlds.

Example 8 *The following formula is a tautology in TL:*

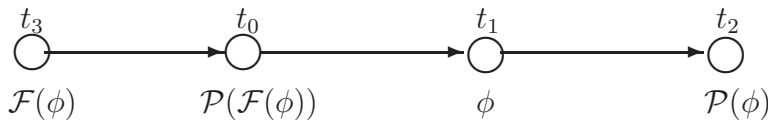
$$\mathcal{F}(\mathcal{P}(\phi)) \rightarrow \mathcal{P}(\mathcal{F}(\phi))$$

Transformation of the condition of the implication yields

$$\pi(\mathcal{F}(\mathcal{P}(\phi))) = (\exists t_2(t_2 \succ t_0 \wedge (\exists t_1(t_1 \prec t_2 \wedge \phi(t_1))))$$

Time is linear in the past, (AX_{4p}), and hence either $t_0 = t_1$, or $t_0 \prec t_1$, or $t_1 \prec t_0$.

Consider case $t_0 \prec t_1$. Since t_0 cannot be the first time point, (AX_{5p}), there is a time point t_3 preceding t_0 .



$\mathcal{F}(\phi)$ holds at t_3 , and hence $\mathcal{P}(\mathcal{F}(\phi))$ holds at t_0 .

It is left as an exercise to prove that in cases $t_0 = t_1$ and $t_1 \prec t_0$, $\mathcal{P}(\mathcal{F}(\phi))$ holds at t_0 .

In all temporal frames for which $\mathcal{F}(\mathcal{P}(\phi))$ is true, $\mathcal{P}(\mathcal{F}(\phi))$ is true also, and hence, the given implication is a tautology.

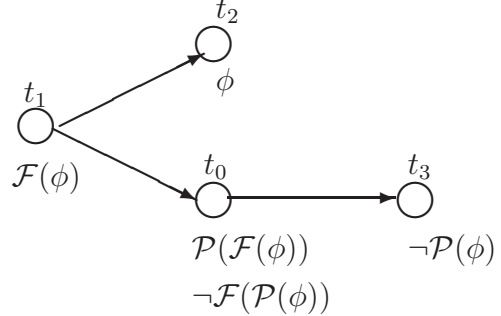
Exercise 9 *Prove that the following expressions are tautologies in TL:*

- $\mathcal{F}(\phi) \leftrightarrow \neg \mathcal{A}_f(\neg \phi)$
- $\mathcal{P}(\phi) \leftrightarrow \neg \mathcal{A}_p(\neg \phi)$

Example 10 The following equivalence is not a tautology in TL: $\mathcal{F}(\mathcal{P}(\phi)) \leftrightarrow \mathcal{P}(\mathcal{F}(\phi))$.

We prove this by constructing a temporal frame M in which $\mathcal{P}(\mathcal{F}(\phi))$ is true, but $\mathcal{F}(\mathcal{P}(\phi))$ is false.

- $\mathcal{T} = \{t_0, t_1, t_2, t_3\}$
- $\prec = \{t_1 \prec t_0, t_0 \prec t_3, t_1 \prec t_2\}$
- $\mathcal{I}(t_0) = \emptyset,$
- $\mathcal{I}(t_1) = \emptyset,$
- $\mathcal{I}(t_2) = \{\phi\},$
- $\mathcal{I}(t_3) = \emptyset$



Recall that this time frame should be understood as the relevant part of a much larger time frame, to be constructed by adding an infinite amount of time points before t_1 and after t_2, t_3 (unbounded past and future) and in between time points (density).

An instance of this proof states that the following two sentences are not equivalent: I will have done it and I was going to do it.

Exercise 11 Prove that $\mathcal{F}(\mathcal{P}(\phi)) \leftrightarrow \mathcal{P}(\mathcal{F}(\phi))$ is a tautology in TL together with $(AX4_f)$, making the future linear instead of branching.

Exercise 12 Consider the following NL sentence: When you have finished your letter, I'll take it to the post office. Show that the following TL-sentence does not capture the intended meaning of the NL-sentence:

$$\mathcal{F}(\mathcal{P}(\text{finish}(\text{you}, \text{letter}))) \rightarrow \mathcal{F}(\text{take_to_postoffice}(\text{I}, \text{letter}))$$

The intended tense of the dependent clause in the sentence is future perfect (will have finished). Write a TL sentence that captures the intended tense.

2.4.3 FOL techniques

Finally, since there exists a whole body of inference techniques and model theoretic proof techniques in FOL, a third possible way to proceed consists in actually transforming a theorem proving problem in TL to FOL and solving it entirely within FOL. This method is particularly suitable for automated theorem proving.

2.5 First-order Modal Temporal Logic

First-order Modal Temporal Logic (FO-MTL) is an extension of propositional MTL allowing universal and existential quantification over the arguments of the predicates. For example, *Someone was born which will be king* is written in FO-MTL as

$$(\exists \mathbf{X})(\mathcal{P}(\text{born}(X)) \wedge \mathcal{F}(\text{king}(X)))$$

Problems arise when the quantified object is created or destroyed over time. For example, consider the following two sentences: *In the past, some people suffered*

from *scurvy* and *Some people (existing today), suffered from scurvy in the past*. These two sentences have a different contents; in the first sentence, we refer to people living in the past who suffered from scurvy at that time; in the second sentence, we refer to people living today who suffered from scurvy at some time in the past. In the first case, the people may already be dead, while in the second case, they are still alive. Hence,

$$\mathcal{P}((\exists X)(\text{scurvy}(X))) \neq (\exists X)(\mathcal{P}(\text{scurvy}(X)))$$

Translation into FOL yields

$$\begin{aligned} (1) \quad \pi(\mathcal{P}((\exists X)(\text{scurvy}(X)))) &= (\exists t_1 \in \mathcal{T})(t_1 \prec t_0 \wedge (\exists X)(\text{scurvy}(X, t_1))) \\ &= (\exists t_1 \in \mathcal{T})(\exists X)(t_1 \prec t_0 \wedge \text{scurvy}(X, t_1)) \end{aligned}$$

$$\begin{aligned} (2) \quad \pi(((\exists X)(\mathcal{P}(\text{scurvy}(X)))) &= (\exists X)(\exists t_1 \in \mathcal{T})(t_1 \prec t_0 \wedge \text{scurvy}(X, t_1)) \\ &= (\exists t_1 \in \mathcal{T})(\exists X)(t_1 \prec t_0 \wedge \text{scurvy}(X, t_1)) \end{aligned}$$

Since (1)=(2), the scheme to transform a (propositional) TL-sentence into FOL is not valid for transforming FO-MTL into FOL. In FOL, the name of an object is assumed to designate that object uniquely at all times. In the example, this is not the case. In general, the existence of objects may change from one point in time to the next. The existence of an object at a certain time can be made explicit by a predicate *exists(X,t)*; *X exists at time point t*. The expressions (1) and (2) can be rewritten correctly as

$$(1) \quad \text{trans}(\mathcal{P}((\exists X)(\text{scurvy}(X)))) = (\exists t_1 \in \mathcal{T})(\exists X)(t_1 \prec t_0 \wedge \text{exists}(X, t_1) \wedge \text{scurvy}(X, t_1))$$

$$(2) \quad \text{trans}((\exists X)(\mathcal{P}(\text{scurvy}(X)))) = (\exists t_1 \in \mathcal{T})(\exists X)(t_1 \prec t_0 \wedge \text{exists}(X, t_0) \wedge \text{scurvy}(X, t_1))$$

Exercise 13 (Optional) Write the following NL-sentence in FO-MTL and in FOL:

There will always jokes be told that were told at a certain time in the past.

References

- [2] M. Fitting. Basic Modal Logic. *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Logical Foundations*. D. M. Gabbay, C. J. Hogger and J. A. Robinson (Eds.). Oxford Science Publications, 1993
- [3] J. F. A. K. Van Benthem. *The logic of time*. Reidel Dordrecht, 1983
- [4] Allan Ramsay. *Formal Methods in AI*. Cambridge University Press, 1988
- [5] Peter Jackson, Han Reichgelt and Frank van Harmelen. *Logic-Based Knowledge Representation*. MIT Press, 1989

3 Situation Calculus

Situation Calculus is a logical theory representing knowledge about dynamic worlds with properties that change over time and with actions that cause these changes. Situation Calculus allows to reason about time and change. Different formalisations of Situation Calculus exist. One well-known approach by Reiter uses classical logic [10]. Our formalisation of Situation Calculus is in ID-logic. A paper on this approach is [9].

Situation calculus is about actions that have effect on time dependent properties. Such time dependent properties will be called *fluents*. Situation calculus, or the simplified version defined in this text, makes a number of simplifying assumption:

- (1) actions have no duration
- (2) there are no simultaneous actions
- (3) actions have immediate effect
- (4) there is no continuous change
- (5) there is an initial time point.

As a consequence, the world presents itself as a sequence of time intervals separated by actions; nothing changes during each interval. Although this clearly is a oversimplification of the world, many problem domains can be modeled in this way. One such an interval will be called a *situation*. It is determined by the linear sequence of *events* or *actions* that led to it¹. We distinguish a situation from a *state*; a state corresponds to a set of properties that can hold at a specific time point². With each situation, we can identify a unique state, namely the set of properties that hold during this situation. But vice versa, it is well-possible that in many different situations, the world is in the same state.

In this setting, an action acts as a state transformer; it transforms one state in a subsequent state through its effect on the properties of the world. The logical representation of what exactly are these effects gives rise to the so called *frame problem*.

There are two ways of representing Situation Calculus in FOL, with a *temporal argument* or with a *reified representation*. In the first approach, a temporal argument representing a situation is present in every fluent of the object language, similar as in the transformation from Tense Logic to FOL. In the reified representation, fluent facts are objects and are denoted by terms. A meta-predicate represents which facts are true in each situation.

At the end of this section , we also -very briefly- discuss some computational problems that can be solved in principle using situation calculus. We only discuss two problems: *prediction problems* through deduction and *planning* problems.

¹In this text, an event is different from an action. An action represents a whole class of events, e.g., the action of *moving block X to location Y*. An event is an action instance, e.g., *the move of block a to the table at time instant 1*.

²Recall that a temporal frame associates a state $I(t)$ to each time point t .

3.1 State Based Representation

Situation Calculus employs time-varying relations or properties and represents which of them are true in different situations separated from each other by events. It treats actions as state transformers. For example, Figure 1 contains a sequence of events which follow the initial state S_0 :

- e_1 : Bob gives book1 to John.
- e_2 : John gives book1 to Mary.
- s_0 : Bob has book1.
John has book2.
Mary has book3.

This results in a sequence of state transitions. Only the properties represented

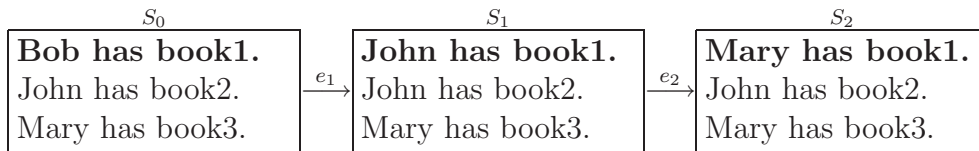


Figure 1: Events as state transformers

in bold change between states; e_1 deletes *Bob has book1* at s_0 and adds *John has book1* at s_1 ; similarly, e_2 deletes *John has book1* and adds *Mary has book1*. The other properties true at s_0 , *John has book2* and *Mary has book3*, are not affected by the events e_1 and e_2 . As a result, they are carried over from s_0 to s_1 and from s_1 to s_2 .

An action transforms one state into a new state. In general, the effect of an action (or event) on a state consists of terminating some properties true in the old state and initiating some new properties false in the old state. All other properties are left unchanged, i.e., remain true if they were true and remain false if they were false.

In practice, an action may initiate and terminate many properties and but in many situations it seemed feasible to represent all these effects in a logic theory. But in [6], McCarthy and Hayes showed that a logic theory should also represent what are *not* effects of an action. For example, one must represent that flipping a specific light switch does not have any effect on the position of objects in the room, on the state of the television, on the books in the cupboard, on music being played, etc, etc. It must also be shown that an action occurring in one situation does only affect the next situation, and not future or past situations. This was in 1969, and at that time, it seemed that an almost incredible amount of FOL axioms was needed to represent exactly what are and what are not the effects of actions. This problem was called the *frame problem*. It has lasted until the late 80-ties and beginning of the 90-ties before correct and general solutions started to appear.

In a Situation Calculus, a formula or set of formulas called the *frame axiom* or the *law of inertia* expresses that an action leaves a property unaltered unless the action explicitly initiates or terminates it.

3.2 Representation of Situation Calculus in ID-logic

The ontology of Situation Calculus consists of actions (e.g. the action of giving an object to some person), situations, fluents (e.g. owning an object), and other sorts of domain dependent objects (e.g. persons, books). We will use also additional predicates to represent the initial state of a fluent and the effects of actions on a fluent.

We will represent the situation calculus in a *sorted* version of FOL extended with (inductive) definitions, i.e., in a sorted version of ID-logic. As we will see, nonmonotone inductive definitions are suitable to handle the frame problem. The use of sorts boils down to splitting up the domain in appropriate subdomains. We use the sorts *Act* and *Sit* to represent actions and situations respectively. Other sorts such as *Person* and *Book* represent domain dependent objects. These sorts are called *object* sorts.

The elements of the ontology will be represented using a vocabulary τ_{Sit} of sorted symbols:

- Objects of object sorts can be represented by constants. E.g. *Bob*, *Mary*, *John* are constants of sort *Person* and represent humans.
- Actions are represented by constants of sort *Act* or by function symbols mapping arguments of object sorts into the sort *Act*. E.g. *Give*(x, y, z) maps tuples of sort (*Person*, *Book*, *Person*) to *Act*. *Give*(*Bob*, *Book1*, *Mary*) represents that Bob gives the book1 to Mary.
- Situations: we distinguish between
 - the initial situation, denoted by the constant S_0 of sort *Sit*
 - successor situations; these are situations that are obtained by applying an action a in a situation s . To represent them, we use a function symbol $Do(a, s)$ mapping tuples of sort (*Act*, *Sit*) into *Sit*. E.g. $Do(Give(John, Book1, Mary), Do(Give(Bob, Book1, John), S_0))$ represents the situation S_2 in Figure 1.
- Fluents: we present two different representations, namely by predicates with *temporal arguments* and a *reified representation*.

A Situation Calculus will consists of several subtheories. In the next sections, we discuss these parts.

3.2.1 Axiomatising the set of situations

We introduced the symbols S_0 and Do to represent situations. Actions are seen as state transformers. For example, if $Give(X, Y, Z)$ denotes the action of X giving Y to Z , then, in Figure 1, $s1$ is the result of applying the action denoted by $Give(Bob, Book1, John)$ in situation $s0$. If S_0 denotes $s0$ then $s1$ is denoted by $Do(Give(Bob, Book1, John), S_0)$ and $s2$ by $Do(Give(John, Book1, Mary), Do(Give(Bob, Book1, John), S_0))$.

What is the set of elements of the sort *Sit*? The set of situations can be defined inductively as the least set containing the initial situation and, for each

situation s and action a , a successor situation obtained by applying a in s . This boils down to the fact that situations correspond to all finite sequences of actions that start in the initial situation. Moreover, two situations that are obtained via a different sequence of actions are considered different.

How can we represent this in ID-logic? The theory consists of two axioms: the Unique Name Axioms $UNA(S_0, Do/2)$ and the Domain Closure Axiom $DCA(S_0, Do/2)$. These axioms will be denoted $UNA(Sit)$ and $DCA(Sit)$:

- $UNA(Sit)$ expresses that different situation terms represent different situations:

$$\left\{ \begin{array}{l} \forall a \forall s \neg (S_0 = Do(a, s)) \\ \forall a \forall a1 \forall s \forall s1 (Do(a, s) = Do(a1, s1) \supset a = a1 \wedge s = s1) \end{array} \right\}$$

- $DCA(Sit)$ is a combination of an inductive definition and an assertion:

$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} Sit(S_0) \leftarrow, \\ \forall a \forall s (Sit(Do(a, s)) \leftarrow Sit(s)) \end{array} \right\} \\ \forall s Sit(s) \end{array} \right\}$$

Notice that these axioms are very similar to the representation of what are the natural numbers. Like the natural numbers, the set of situations forms a well-founded order under the precedence relation. We will exploit this by defining fluents using nonmonotone induction over the precedence relation.

There are two important remarks.

1. Note that these axioms $UNA(Sit)$ and $DCA(Sit)$ do not fully determine the set of situations. In fact, the set of situations depends on what is the set of actions and is fully determined by it. Another subtheory of situation calculus may serve to axiomatise what are the set of actions.
2. It is also important to notice that although each sequence of actions corresponds a situation term and that in each structure I , such a term denotes an object in the domain of Sit , many such terms and the denoted situation objects do not correspond to situations that could exist in reality. For example, the term $Do(Give(John, Book1, Bob), S_0)$ does not represent a situation that can occur in reality, for the simple reason that the action $Give(John, Book1, Bob)$ is impossible in the initial situation. Indeed, John does not possess book1 so he cannot give it away. Other examples of situation terms denoting impossible situations are:

- (1) $Do(Give(John, Book2, Mary), Do(Give(John, Book1, Bob), S_0))$: each successor situation of an impossible situation is impossible as well.
- (2) $Do(Give(Bob, Book1, Mary), Do(Give(Bob, Book1, Mary), S_0))$

Not all actions can be applied in each situation. This basically means that Do denotes a *partial function*, not a total function. In the course, we have seen how to simulate partial functions. Another part of a situation calculus will be concerned with expressing which situations are really possible.

The set of actions can be formalised along similar lines. One needs to add $UNA(Act)$ to express that different action terms represent different actions. Often one will include $DCA(Act)$ to represent that the only actions are those denoted by action terms.

3.2.2 Frame axioms

The method of temporal arguments (TA) represents fluents as predicates with an extra situation argument of sort *Sit*. For example, we could use the fluent predicate $owns(x, y, s)$ of sort $(Person, Book, Sit)$ to denote the property that x owns book y in situation s .

In general a fluent can be represented as a predicate symbol with exactly 1 argument of sort *Sit* and all other arguments are of object sorts. There are no action arguments.

Now we discuss how to represent the dynamics of the world. Consider the following FOL theory:

$$\left\{ \begin{array}{l} Owns(Bob, Book1, S_0) \\ Owns(John, Book2, S_0) \\ Owns(Mary, Book3, S_0) \\ \forall x \forall y \forall z \forall s Owns(z, y, Do(Give(x, y, z), s)) \\ \forall x \forall y \forall z \forall s \neg Owns(x, y, Do(Give(x, y, z), s)) \end{array} \right\}$$

Each of these FOL axioms represents a basic property of the domain. The first three atomic formulas represent the initial situation s_0 of Figure 1. The two last rules represents that initiating and terminating effects of action *Give* of the book's ownership by the receiver, respectively the donator.

The problem with this FOL theory is that it does not represent what are not effects of the *Give* action. Consequently, it is impossible to prove from this theory that book2 and book3 do not change from owner during situations $s_0 - s_2$. Or, this theory does not entail:

$$\begin{array}{l} Owns(Mary, Book3, Do(Give(John, Mary, Book1), \\ Do(Give(Bob, John, Book1), S_0))) \end{array}$$

Actions typically affect only a very small part of the world. The rest of the world remains unchanged and in FOL this should also be represented. This is the aforementioned *frame problem* which was addressed in [6] by McCarthy and Hayes in 69. It was thought then that classical logic was unable to express this myriad of other facts in a concise way.

The paper of McCarthy and Hayes was an important turning point for logic-based AI. The FOL theorem proving project of the sixties failed to bring up theorem provers that could be of use to solve practical problems. Now, the frame problem showed serious problems at the representational level of classical logic. This paper was the start of research on how to represent *common sense knowledge* as opposed to mathematical logic for which FOL had proven very successful. In this new area, so-called *nonmonotonic logics* were developed for representing common sense knowledge such as defaults. In the context of actions, the hope was to solve the frame problem by a default expressing the following:

Normally, an action does not change a property in the world, unless explicitly stated.

The solution that we give for the frame problem does not use defaults but an inductive definition. We define all fluents by simultaneous, and in general, non-monotone induction over the well-founded precedence order of situations.

Below, \mathbf{x} is used to denote a tuple (x_1, \dots, x_n) of arguments of object sorts. For each fluent $P(\mathbf{x}, s)$, we introduce three new predicates:

- $Initially_P(\mathbf{x})$: $P(\mathbf{x})$ is initially true. E.g. $Initially_{Owns}$ has sort $(Person, Book)$.
- $Initiates_P(a, \mathbf{x}, s)$: action a causes fluent $P(\mathbf{x})$ to become true in situation s . E.g. $Initiates_{Owns}$ has sort $(Act, Person, Book, Sit)$.
- $Terminates_P(a, \mathbf{x}, s)$: action a causes $P(\mathbf{x})$ to become false in situation s . E.g. $Terminates_{Owns}$ has sort $(Act, Person, Book, Sit)$.

The key part of our Situation Calculus in ID-logic is a large definition Δ_{sc} defining all fluents P and their effect predicates $Initiates_P$ and $Terminates_P$. It contains the following type of rules:

The frame rules, for each fluent P :

$$\begin{aligned} \forall \mathbf{x}(P(\mathbf{x}, S_0) \leftarrow Initially_P(\mathbf{x})) \\ \forall a \forall s \forall \mathbf{x}(P(\mathbf{x}, Do(a, s)) \leftarrow Initiates_P(a, \mathbf{x}, s)) \\ \forall a \forall s \forall \mathbf{x}(P(\mathbf{x}, Do(a, s)) \leftarrow P(\mathbf{x}, s) \wedge \neg Terminates_P(a, \mathbf{x}, s)) \end{aligned}$$

The first rule formalises the fluent in the initial state. The second rule specifies that if an action a initiates a fluent in a situation s , then this fluent will be true in the successor situation $Do(a, s)$. The last rule expresses that if a fluent is true in situation s and action a does not terminate it, then the fluent continues to be true in the successor situation $Do(a, s)$. This rule can be considered to be the *law of inertia*. Here it is a definitional rule.

effect rules: The set of above definitional rules (three per predicate) have to be completed with domain dependent rules representing the effect of actions. These rules are the definitional rules of $Initiates_P$ and $Terminates_P$.

For each initiating effect of an action A on fluent P , there should be a rule:

$$\forall \mathbf{t} \forall s(Initiates_P(A, \mathbf{t}, s) \leftarrow \Psi[s])$$

Here Ψ is a formula representing the precondition for this effect. An example is:

$$\forall x \forall y \forall b \forall s(Initiates_{Owns}(Give(x, b, y), y, b, s) \leftarrow owns(x, b, s))$$

Analogously, there should be one rule one rule per terminating effect of an action A on fluent P :

$$\forall \mathbf{t} \forall s(Terminates_P(A, \mathbf{t}, s) \leftarrow \Psi[s])$$

Also here Ψ represents the precondition of the effect. An example is

$$\forall x \forall y \forall b \forall s (Terminates_{Owns}(Give(x, b, y), x, b, s) \leftarrow x \neq y)$$

We summarize the definition Δ_{sc} for the fluent and action of Figure 1:

$$\left\{ \begin{array}{l} \forall p \forall b (Owns(p, b, S_0) \leftarrow Initially_{Owns}(p, b)), \\ \forall a \forall s \forall p \forall b (Owns(p, b, Do(a, s)) \leftarrow Initiates_{Owns}(a, p, b, s)), \\ \forall a \forall s \forall p \forall b (Owns(p, b, Do(a, s)) \leftarrow Owns(p, b, s) \wedge \\ \qquad \qquad \qquad \neg Terminates_{Owns}(a, p, b, s)), \\ \forall x \forall y \forall b \forall s (Initiates_{Owns}(Give(x, b, y), y, b, s) \leftarrow Owns(x, b, s)) \\ \forall x \forall y \forall b \forall s (Terminates_{Owns}(Give(x, b, y), x, b, s) \leftarrow x \neq y) \end{array} \right\}$$

We end this presentation with two important remarks.

1. The example Figure 1 is a simple one with only one fluent *Owns* and one action *Give*. In general there may be many actions and fluents and many effect rules. Together they constitute one big definition. This definition is:

- **Inductive:** the law of inertia is clearly an inductive rule. Another level of induction is that fluent predicates are defined in terms of effect predicates, and effect predicates are defined in terms of fluent predicates.
- **Non-monotone induction:** the body of the inertia law contains the negative literal $\neg Terminates_P(a, \mathbf{x}, s)$. If an effect rule defines this effect predicate in terms of fluent P , then we have a clear case of non-monotone induction.

An example is a representation of flipping a switch of a light bulb. We use a fluent $On(s)$ to represent that the light is on, and an action $Flip$ to represent that the switch of the bulb is flipped. This is the corresponding Δ_{sc} :

$$\left\{ \begin{array}{l} On(S_0) \leftarrow Initially_{On}, \\ \forall a \forall s (On(Do(a, s)) \leftarrow Initiates_{On}(a, s)), \\ \forall a \forall s (On(Do(a, s)) \leftarrow On(s) \wedge \neg Terminates_{On}(a, s)), \\ \forall s (Initiates_{On}(Do(Flip, s)) \leftarrow \neg On(s)), \\ \forall s (Terminates_{On}(Do(Flip, s)) \leftarrow On(s)) \end{array} \right\}$$

This is an induction over the well-founded precedence order between situations. This order has minimal element S_0 . It is easy to see that the rules define fluents in successor situations in terms of their predecessors.

2. Notice that the frame axioms are asymmetric with respect to truth and falsity of fluents. For example, the law of inertia only represents half of the inertia property explicitly. The second half of this law is that *if a fluent is false and is not initiated by action a in situation s, then it remains false*. This property can be represented by the following FOL implication:

$$\forall \mathbf{x} \forall a \forall s (\neg P(\mathbf{x}, Do(a, s)) \subset \neg P(\mathbf{x}, s) \wedge \neg Initiates_p(a, \mathbf{x}, s))$$

Another law that is not explicitly represented is that if an action terminates a property, then it is false in the successor state:

$$\forall \mathbf{x} \forall a \forall s (\neg P(\mathbf{x}, Do(a, s)) \subset Terminates_p(a, \mathbf{x}, s))$$

Notice that these rules cannot be definitional rules due to the negation in front. In fact, we do not need to add these rules. This is an illustration of a basic principle of definitions, namely that in a definition, you only represent the rules deriving when the defined predicates are *true*. By the principle of inductive definition, in all other cases, the defined predicates are assumed to be false and this subsumes any implication that explicitly derives falsity of a defined predicate.

Here, it can be shown that both rules are logically implied by the definition Δ_{sc} at least if the following natural requirement is satisfied:

$$\forall a \forall \mathbf{x} \forall s (\neg Initiates_p(a, \mathbf{x}, s) \vee \neg Terminates_p(a, \mathbf{x}, s))$$

This is the requirement that an action does not cause P to become true and to become false in the same situation.

3.2.3 Formalising the initial situation

The initial situation can be formalised in different ways. In the example in Figure 1 we have complete knowledge about the initial situation in the scenario. Therefore, we can define the predicate $Initially_{Owns}$: Figure 1:

$$\left\{ \begin{array}{l} Initially_{Owns}(Bob, Book1) \leftarrow, \\ Initially_{Owns}(John, Book2) \leftarrow, \\ Initially_{Owns}(Mary, Book3) \leftarrow \end{array} \right\}$$

When we do not have complete knowledge on the initial situation, then we can use FOL assertions. For example, the following axiom represents that Mary has book3 or book2:

$$Initially_{Owns}(Mary, Book2) \vee Initially_{Owns}(Mary, Book3)$$

In general, we must add more axioms to formalise what is known about the object sorts. In the example of Figure 1, we must express $UNA(Person)$ and $UNA(Book)$. If we have complete knowledge about what are persons and/or what are books, we can add $DCA(Person)$ and/or $DCA(Book)$.

Exercise 14 We could represent that in the initial situation, John has book1 or Mary has book1 in two ways: by the FOL assertion

$$Initially_{Owns}(John, Book1) \vee Initially_{Owns}(Mary, Book1)$$

or by

$$Owns(John, Book1, S_0) \vee Owns(Mary, Book1, S_0)$$

Argue why this is equivalent.

3.2.4 Action preconditions

Actions cannot occur in all circumstances. E.g. a robot can pick up an object only if the robot is free, the object is a block and it is clear. Basically, this means that in such circumstances, the *Do* function symbol is used to denote a *partial function*.

Partial functions can be simulated in logic by using a predicate which filters out the “existing” objects from the unexisting ones. Here we apply this technique.

In situation calculus, we simulate this by defining a filter predicate $Possible(s)$ of sort Sit which indicates which situations are obtained via a sequence of actions whose preconditions were satisfied.

$$\left\{ \begin{array}{l} Possible(S_0) \leftarrow \\ Possible(Do(a, s)) \leftarrow Poss(a, s) \wedge Possible(s) \end{array} \right\}$$

We define the predicate $Poss$ of sort (Act, Sit) . An SC theory contains a definition for $Poss$ consisting of rules of the form $\forall s(Poss(A, s) \leftarrow \Psi)$ where Ψ represents a sufficient precondition for action A to be executable. E.g. in the block world:

$$\left\{ \begin{array}{l} \forall x \forall s (Poss(Pick(x), s)) \leftarrow Block(x) \wedge Holds(Free_robot, s) \wedge \\ \qquad \qquad \qquad Holds(Clear(x), s) \\ \forall x \forall y \forall s (Poss(Put(x, y), s)) \leftarrow Block(x) \wedge Location(y) \wedge x \neq y \wedge \\ \qquad \qquad \qquad Holds(Clasped(x), s) \wedge \\ \qquad \qquad \qquad Holds(Clear(y), s) \end{array} \right\}$$

The *possible* situations are those for which $Possible(s)$ holds.

3.2.5 Summary: TA Situation Calculus

A Situation calculus T_{sc} consists of the following components, i.e. $T_{sc} = T_{Sit} \cup T_{Act} \cup \{\Delta_{sc}\} \cup T_{ini} \cup T_{pre}$:

- (1) $T_{Sit} = UNA(Sit) \cup DCA(Sit)$: axiomatisation of situations
- (2) T_{Act} : axiomatisation of situations
- (3) $\{\Delta_{sc}\}$: the situation calculus definition
- (4) T_{ini} : the theory of initial situation and of object sorts
- (5) T_{pre} : the action precondition theory consisting of definitions of $Possible(s)$ and $Poss(a, s)$.

3.2.6 The reified solution

In the *reified representation* (TR) of Situation Calculus, we introduce a new sort *Fluent*. All fluents are represented by function symbols of sort *Fluent* with arguments of object sorts. E.g. in this solution, the fluent *Owns* is a function symbol

of sort $(Person, Book) : Fluent$. Fluent terms such as $Owns(Mary, Book1)$ represent facts whose truth may change over time. The theory $UNA(Fluent)$ must be added.

A meta-predicate $Holds(p, s)$ of sort $(Fluent, Sit)$ is used to represent which fluent facts p are true in situation s . In this representation, we use additional predicate symbols $Initially(p)$ of sort $Fluent$, and $Initiates(a, p, s)$ and $Terminates(a, p, s)$ of sort $(Act, Fluent, Sit)$. In the case of the book example, is represented in this representation is:

$$\left\{ \begin{array}{l} \forall p(Holds(p, S_0) \leftarrow Initially(p)), \\ \forall a \forall s \forall p(Holds(p, Do(a, s)) \leftarrow Initiates(a, p, s)), \\ \forall a \forall s \forall p(Holds(p, Do(a, s)) \leftarrow Holds(p, s) \wedge \\ \qquad \qquad \qquad \neg Terminates(a, p, s)), \\ \forall x \forall y \forall b \forall s(Initiates(Give(x, b, y), Owns(y, b), s) \leftarrow Owns(x, b, s)) \\ \forall x \forall y \forall b \forall s(Terminates(Give(x, b, y), Owns(x, b), s) \leftarrow x \neq y) \end{array} \right\}$$

The initial situation could be represented by the definition:

$$\left\{ \begin{array}{l} Initially(Owns(Bob, Book1)) \leftarrow, \\ Initially(Owns(John, Book2)) \leftarrow, \\ Initially(Owns(Mary, Book3)) \leftarrow \end{array} \right\}$$

TR is more expressive than TA because it allows quantifying over properties in the object language. As a result, TR can express general rules relating the effect of actions to the properties of any problem domain, whereas in TA, these rules must be formulated for every particular problem domain. In particular, TR contains only 3 frame axioms while TA contains 3 frame axioms per fluent.

3.2.7 A block world example

This is the standard first example in AI-planning. The blocks world is a closed world containing blocks and one robot manipulator to pick up and put down blocks on a table or on other blocks. To pick up a block, the block must be clear and the robot must be free.

To represent it, we use following ontology and vocabulary:

- One object sort: Loc (location containing table and blocks) with predicate $Block(b)$ to represent the blocks
- Fluents:
 - $On(x, y)$ of sort (Loc, Loc) : x is on y
 - $Clear(x)$ of sort (Loc) : something can be put on x
 - $Clasped(x)$ of sort (Loc) : the robot clasps block x
 - $Free_robot$: the robot does not clasp any object
- Actions:
 - $Put(x, y)$ of sort (Loc, Loc) : the robot puts block x on location y

- $Pick(x)$ of sort (Loc): the robot picks up block x

We first present the defining rules of *Initiates* and *Terminates*. These should be added to the 3 frame axioms:

$$\begin{aligned} \forall x \forall y \forall s \text{Initiates}(Put(x, y), On(x, y), s) &\leftarrow \\ \forall x \forall y \forall s \text{Initiates}(Put(x, y), Clear(x), s) &\leftarrow \\ \forall x \forall y \forall s \text{Initiates}(Put(x, y), Free_robot, s) &\leftarrow \\ \forall x \forall y \forall s (\text{Initiates}(Pick(y), Clear(x), s) &\leftarrow Holds(on(y, x), s) \wedge Block(x)) \\ \forall x \forall s \text{Initiates}(Pick(x), Clasp(ed)(x), s) &\leftarrow \end{aligned}$$

$$\begin{aligned} \forall x \forall y \forall s (\text{Terminates}(Put(x, y), Clear(y), s) &\leftarrow Block(y)) \\ \forall x \forall y \forall s \text{Terminates}(Put(x, y), Clasp(ed)(x), s) &\leftarrow \\ \forall x \forall s \text{Terminates}(Pick(x), Clear(x), s) &\leftarrow \\ \forall x \forall y \forall s \text{Terminates}(Pick(x), On(x, y), s) &\leftarrow \\ \forall x \forall y \forall s \text{Terminates}(Pick(x), Free_robot, s) &\leftarrow \end{aligned}$$

Notice that only blocks can be picked up. Also, putting something on the table does not delete that it is clear. We should add that the table is clear in the initial situation. Since no actions can terminate this, the table remains clear.

3.2.8 Expressiveness of situation calculus

The expressiveness of a language is determined by its capacity to represent objects, events and relations precisely and correctly.

Topology of time The axiomatisation characteristics of Situation Calculus determine the time topology, independent of the Situation Calculus' representation.

- **point-based:** Although situations can be thought of as interval of time points in between two actions, there is no change allowed during these intervals. There is no support for reasoning about overlapping intervals. As a consequence, Situation Calculus cannot express duration-related properties such as *John was sleeping while Mary was being kidnapped*.
- **Linear past, branching future:** In each model of T_{sit} the set of situations under the precedence relation satisfies the axioms of linear past. The future is branching, in the sense that in each situation, each action starts a different future.
- **bounded in the past and unbounded in the future:** The initial situation S_0 is the lower bound of this time line, and new events can create new situations indefinitely.
- **discrete time:** Events are state transitions. Situation Calculus only expresses what is true before and after the transition, but not during the transition.

Context-dependent effects The effects of a context-dependent action depends upon the properties that are true in the situation when the action is applied. For example, turning a bottle upside down would wet the table only if the bottle contains liquid and only if the bottle is open.

$$\begin{aligned} \forall x \forall s (Initiates(Turn(x), Empty(x), s) \leftarrow & Bottle(x) \wedge Holds(Open(x), s)) \\ \forall x \forall t \forall s (Initiates(Turn(x), Wet(t), s) \leftarrow & Bottle(x) \wedge Holds(At(x, t), s) \wedge \\ & Holds(Open(x), s) \wedge \\ & Holds(Filled(x), s)) \\ \forall x \forall s (Terminates(Turn(x), Filled(x), s) \leftarrow & Bottle(x) \wedge Holds(Open(x), s)) \end{aligned}$$

Defining dynamic properties Sometimes fluents can be *defined* in terms of other fluents. We can see two cases in the blocks world:

- *Clear(b)* can be defined in terms of *On*. The defining rule is:

$$\forall b \forall s Holds(Clear(b), s) \leftarrow \neg \exists y Holds(On(y, b), s)$$

- *Free_robot* can be defined in terms of *Clasped*. The defining rule is:

$$\forall s Holds(Free_robot, s) \leftarrow \neg \exists y Holds(Clasped(y), s)$$

It requires some care to add the rules defining *Holds* in case of defined fluents to Δ_{sc} . First, we should delete the effect rules of the defined fluents and replace them by the defining rules. By introducing definitional rules of *Clear* and *Free_robot*, the number of effect rules is reduced to 4 instead of 8:

$$\begin{aligned} \forall x \forall y \forall s Initiates(Put(x, y), On(x, y), s) \leftarrow \\ \forall x \forall s Initiates(Pick(x), Clasped(x), s) \leftarrow \\ \forall x \forall y \forall s Terminates(Put(x, y), Clasped(x), s) \leftarrow \\ \forall x \forall y \forall s Terminates(Pick(x), On(x, y), s) \leftarrow \end{aligned}$$

There is a second problem. Consider law of inertia in case of defined property. E.g.

$$\forall a \forall s (Holds(Free_robot, Do(a, s)) \leftarrow Holds(Free_robot, s) \wedge \neg Terminates(a, Free_robot, s))$$

Since no explicit terminating effects are given for *free_robot*, the inertia rule implies that once *free_robot* becomes true, it remains true forever.

The solution of this problem is to restrict the application of the frame axioms to fluents which are not defined. To this end, we distinguish between *primitive* (or *frame*) fluents and *defined* (or *derived*) fluents and introduce a predicate symbol *Frame* of sort *Fluent* which is defined here as:

$$\left\{ \begin{array}{l} \forall x \forall y Frame(On(x, y)) \leftarrow \\ \forall x Frame(Clasped(x)) \leftarrow \end{array} \right\}$$

We restrict application of the frame axioms to primitive fluents.

$$\begin{aligned} \forall p(Holds(p, S_0) &\leftarrow Frame(p) \wedge Initially(p)) \\ \forall p \forall a \forall s(Holds(p, Do(a, s)) &\leftarrow Frame(p) \wedge Initiates(a, p, s)) \\ \forall p \forall a \forall s(Holds(p, Do(a, s)) &\leftarrow Frame(p) \wedge Holds(p, s) \wedge \\ &\quad \neg Terminates(a, p, s)) \end{aligned}$$

These rules together with the defining rules for *Free_robot* and *Clear(b)* and the effect rules of the actions on the frame fluents, form a correct definition Δ_{sc} for the block world problem.

In the TR representation (with temporal arguments), the problem does not arise because there all frame axioms work only for one predicate at the time. E.g. we would define:

$$Clear(b, s) \leftarrow \neg \exists y. On(y, b, s)$$

Interestingly, we can also define fluents by induction. For example assume we need the transitive closure of the predicate *On* in blocks world and represent this relation by *TOn*. The definitional rules to be included in Δ_{sc} are:

$$\begin{aligned} \forall x \forall y \forall s(Holds(TOn(x, y), s) &\leftarrow Holds(On(x, y), s)) \\ \forall x \forall y \forall z \forall s(Holds(TOn(x, y), s) &\leftarrow Holds(TOn(x, z), s) \wedge \\ &\quad Holds(TOn(z, y), s)) \end{aligned}$$

Notice that we mix monotone and non-monotone induction. This is an iterated inductive definition which can be expressed in ID-logic. On the other hand, expressing it in other logics such as FOL or even Second Order Logic is a hard task.

Other extensions Depending on the problem domain and the problems that must be solved, Situation Calculus must be extended with other features:

- numerical time: sometimes it is necessary to know the numerical time of a situation
- undeterministic effects: e.g. rolling a dice
- simultaneous actions
- actions with duration: e.g. baking a bread lasts 1h
- continuous change: moving from one place to another
- some actions do not change the world but change an agents knowledge of the world: sensing actions. E.g. sensors on mobile robots, our eyes and ears. Representing sensing actions requires that the state at a situation does not only include objective facts about the world but also the knowledge of the agent who can perform sensing actions. Sensing actions modify this knowledge.
- compound actions: actions that consists of subactions

• ...

Exercise 15 Assume that there is a fluent content of sort (*Bottle, real*) to represent the content of a bottle. Assume there is a time independent predicate *capacity(b, c)* of sort (*bottle, real*) to indicate the capacity of a bottle. Define 3 fluents signifying respectively that a bottle is empty, a bottle is filled and a bottle is full.

Exercise 16 (optional) An action has a non-deterministic effect if some of its effects are possible but not certain. An example is rolling a dice, which may lead to an outcome of one number between 1 and 6. Another example is the spinning of a gun chamber in Russian Roulette: this spinning may load or unload the gun (after which the player shoots in his head).

Think of an extension of situation calculus in ID-logic to represent such non-deterministic effects. (Hint: introduce an open predicate(*s*) which captures the non-determinism of the action and use this open predicate in the effect rules. For example, in the Russian Roulette example, we might use a predicate *good_luck(s)* which represents that in situation *s*, the player has good luck and spinning the gun chamber will unload the gun.)

Exercise 17 (Optional, difficult) A compound action is an action that can be decomposed into smaller subactions. For example, the action of making a hole in the wall can be decomposed into grasping a drill, positioning the drill on the wall, and drilling a hole in the wall. Similarly, the action drilling a hole in the wall can be further decomposed: connect the drill's plug in the outlet, turn on the drill, drill, turn off the drill, disconnect the drill's plug. The granularity of representation determines the atomic events. A compound action is a way to abstract the representation of an action.

How would you represent compound actions in Situation Calculus and calculate with them? Demonstrate your method for the hole-drilling example.

Exercise 18 In the bottle pouring example, why didn't we wrap the property *bottle(X)* with the predicate *Holds* as follows?

$$\begin{aligned} \forall x \forall s (\text{Initiates}(\text{Turn}(x), \text{Empty}(x), s) \leftarrow & \text{holds}(\text{bottle}(\mathbf{x}), \mathbf{s}) \wedge \\ & \text{Holds}(\text{Open}(x), s)) \\ \forall x \forall t \forall s (\text{Initiates}(\text{Turn}(x), \text{Wet}(t), s) \leftarrow & \mathbf{Holds}(\mathbf{Bottle}(\mathbf{x}), \mathbf{s}) \wedge \\ & \text{Holds}(\text{At}(x, t), s) \wedge \\ & \text{Holds}(\text{Open}(x), s) \wedge \\ & \text{Holds}(\text{Filled}(x), s)) \\ \forall x \forall s (\text{Terminates}(\text{Turn}(x), \text{Filled}(x), s) \leftarrow & \mathbf{Holds}(\mathbf{Bottle}(\mathbf{x}), \mathbf{s}) \wedge \\ & \text{Holds}(\text{Open}(x), s)) \end{aligned}$$

3.3 Temporal Reasoning with Situation Calculus

A general theme of the course Knowledge Representation is that to perform tasks, or to write algorithms or procedures that solve problems, humans have declarative, non-procedural, knowledge about the world and rely on it when performing

the task or writing the procedure. Such knowledge in the current state of software technology is almost never explicitated. The main focus of this course was to study this declarative knowledge and how to represent it. A second goal was to show how this knowledge is used when solving computational tasks.

In the previous section, we presented Situation Calculus as a simple formalism for representing declarative knowledge about actions and fluents. We now discuss some types of reasoning problems that can be solved (in principle) using a Situation Calculus.

3.4 Situation Calculus in Prolog

Logic programming, or the pure subset of Prolog, can be understood as a sublogic of ID-logic, including *DCA*, *UNA*, and one definition defining all predicates; moreover bodies of definitional rules contain only conjunction \wedge and disjunction \vee and each occurrence of the negation symbol \neg is in front of an atom $\neg A$. There are no other FOL assertions. Such theories represent complete knowledge.

This means that if we have complete knowledge about all predicates, we can translate a situation calculus into a Prolog program. This is the case in the book example of Figure 1 and basically in all situation calculi with complete knowledge on initial state and object sorts.

Prolog uses different notational conventions than FOL and ID-logic. Here are two examples:

$$\begin{array}{l} \forall x \forall y (P(A, x) \leftarrow (Q(x) \wedge (Q(y) \vee P(y)))) \longrightarrow \\ \qquad P(a, X) \text{ :- } Q(a) \text{ , } (Q(X) \mid P(X, a)) . \\ \forall x \forall y (P(A, x) \leftarrow \qquad \qquad \qquad \longrightarrow \\ \qquad P(a, X) . \end{array}$$

We summarize the transformations:

	\leftarrow	\longrightarrow	 :-
(1) logical symbols have a different notation:	\wedge	\longrightarrow	 " , "
	\vee	\longrightarrow	\mid
	\neg	\longrightarrow	 not

(2) Prolog variables start with capitals; Prolog constants start with minuscules

(3) a prolog rule ends with “.”

(4) an atomic rule $\forall x P(A, x) \leftarrow$ is written as $P(a, X)$.

(5) *DCA* and *UNA* are implicit and can be dropped.

Applying these rules on the book example theory yields:

```
Owns(P,B,s_0) :- Initially_Owns(P,B)).
Owns(P,B,Do(A,S)) :- Initiates_Owns(A,P,B,S)).
Owns(P,B,Do(A,S)) :- Owns(P,B,S), not Terminates_Owns(A,P,B,S).
Initiates_Owns(Give(X,B,Y),Y,B,S) :- Owns(X,B,S).
Terminates_Owns(Give(X,B,Y),X,B,S) :- not X = Y.
```

```
Initially_Owns(bob,book1).
Initially_Owns(john,book2).
Initially_Owns(mary,book3).
```

We can now use prolog to query this theory. For example, we may ask the prolog query

```
?-Owns(P,Book1,Do(Give(john,mary,book1),Do(Give(bob,john,book1),s_0))).
```

Prolog will find the person that owns book1 in the given situation.

```
Yes P=mary
```

We could in principle also ask for all situations in which Mary owns book1 by the query:

```
?-Owns(Mary,Book1,S).
```

This query has an infinite number of solutions. If lucky Prolog will loop, generating the solutions one by one for indefinite time. Unfortunately, most Prolog systems will end in failure or produce erroneous answers³.

Observe that basically, we can understand such a query with situation variable as a query of a goal situation in which certain properties are true. This is a planning problem. This type of problems are very interesting in AI, and this makes it double regrettable that Prolog cannot handle such problems.

3.5 Prediction, Postdiction

The prediction problem in Temporal Reasoning is about deriving the causal consequences of a given set of events in a future time point. For example, in Figure 1, from the occurrence of the events $e1$ and $e2$, together with a description of the initial situation $s0$, it is possible to derive the properties of the resulting situation $s2$. In a Situation Calculus with complete knowledge on initial situation, prediction can be automated using the execution mechanism of Prolog. This was illustrated in the previous section.

Prediction problems may also arise in the context of incomplete knowledge on the initial situation. For example assume in the initial situation, there are two packets. Both make a ticking noise and we know that one of them is an alarm clock and the other a bomb without knowing which one. The action of throwing a bomb in a toilet unloads the bomb. The effects can be represented by the definitional rules:

$$\begin{aligned} &\forall x \forall s \text{Terminates}(\text{Throw}(x), \text{Loaded}(x), s) \\ &\forall x \forall s (\text{Terminates}(\text{Explode}, \text{Alive}, s) \leftarrow \exists p (\text{Bomb}(p) \wedge \text{Holds}(\text{Loaded}(x), s))) \end{aligned}$$

³The technical reason is a floundering negation problem: at some point, Prolog might try to use the inertia law which will lead to the goal `not Terminates_Owns(A,mary,book1,S)`. This goal contains free variables inside negation, which cannot be handled correctly by Prolog.

The knowledge about the initial state is represented as FOL assertions:

$$\begin{aligned} &(\forall x \text{Bomb}(x) \supset x = P1 \vee x = P2) \\ &\text{bomb}(P1) \equiv \neg \text{Bomb}(P2) \\ &\forall p(\text{Initially}(\text{Loaded}(p)) \equiv \text{Bomb}(p)) \\ &\text{Initially}(\text{Alive}) \end{aligned}$$

Notice that the resulting ID-logic theory cannot be mapped to Prolog. This theory does not allow to do definite predictions. For example, it is impossible to prove or disprove that

$$\text{Holds}(\text{Alive}, \text{Do}(\text{Explode}, \text{Do}(\text{Throw}(P1), S_0)))$$

On the other hand, the following formula is provably true

$$\text{Holds}(\text{Alive}, \text{Do}(\text{Explode}, \text{Do}(\text{Throw}(P2), \text{Do}(\text{Throw}(P1), S_0))))$$

Prediction in the context of incomplete information on the past is sometimes called *ambiguous prediction*. Abductive reasoning may be useful to solve ambiguous prediction problems. For example abductive reasoning on the query

$$\text{Holds}(\text{Alive}, \text{Do}(\text{Explode}, \text{Do}(\text{Throw}(P1), S_0)))$$

would return the hypothesis

$$\text{Bomb}(P1) \wedge \neg \text{Bomb}(P2)$$

The inverse sort of reasoning is Postdiction. In postdiction problems, some given final situation is found which is the result of a partially known sequence of events in a partially known initial situation. The postdiction problem is to reason about the initial situation and the unknown actions that explain the observations of the final situation. An example would be the solving of some murder mystery. Abductive reasoning is a very useful form of reasoning for Postdiction problems. The A-system [11] is an example of an abductive system that can be used to solve this type of problems.

It is inherent to Postdiction problems that the input knowledge is incomplete on either the initial situation or the sequence of actions or both. Notice that an AI-planning problem can be considered as a special postdiction problem, with known initial situation but unknown sequence of actions. On the other hand, we may also have problems where we exactly know what events took place but not the initial situation.

Exercise 19 *The following scenario is known as the Yale Turkey Shooting problem.⁴ Fred is a turkey which at any time is either dead or alive. There is also a gun which is either loaded or unloaded. Initially, Fred is alive. The gun becomes loaded at any time a load event happens. When a turkey is shot with a loaded gun, it gets killed. There is also an action wait with no effect. The following three events are executed consecutively starting from s0: load, wait and shoot.*

Present the Yale Turkey Shooting problem in Situation Calculus. Program and execute it in Prolog.

⁴Hanks, S. and McDermott, D., "Default Reasoning, Nonmonotonic Logic, and the Frame Problem", *Proceedings of the National Conference on Artificial Intelligence, Philadelphia, PA, 1986*, pp. 328-333

Exercise 20 A variant postdiction problem is the following murder mystery. Take the Yale Turkey Shooting except that the initial situation is unknown and we observe that after actions of waiting and shooting, Fred is dead. Represent this knowledge. The postdiction problem is to find possible explanations on the initial situation that can explain this. What are the possible explanations? (There are two of them.)

3.6 Planning

Planning is an explanation problem; it tries to explain why certain properties are true at certain times. Planning has received attention in Artificial Intelligence (AI) because it is necessary for intelligent behavior. Its automation is important for future applications in manufacturing, robotics, navigation, management, etc.

Given a description of possible actions and how each of these actions changes the state of affairs, a planner tries to find which actions, together with their ordering in time, achieve a given *goal* starting from an initial state of the world. In what is called AI-planning, the planning problem is restricted to *domain-independent preplanning*; the techniques used to generate a plan are independent of the problem domain, and the plan is constructed prior to its execution. The planner constructs the plan in a *closed world* in which all relevant actions and properties are known and complete knowledge on initial situation is given. During its execution, no events external to the plan can change this world.

One of the earliest and best known systems for AI-planning is the strips planner. The input of a strips planner is a formal description of a planning domain, which roughly contains the same components and describes the same type of information as in situation calculus: complete knowledge on initial situation, initiating and terminating effects of actions, preconditions of actions and goals. Consequently, the strips-planner language should be understood as a special purpose formal logic for representing planning domains, with its own syntax, and a number of shorthand notations. Compared to situation calculus, it imposes strong limitations (e.g. in early versions of the strips language, no conditional effects are allowed). These limitations make it easier to develop faster planning systems.

A number of automated planning systems have been developed that use Situation Calculus as their theory of time. In the standard setting of AI-planning, there is complete knowledge on the initial situation. Consequently, a domain theory in Situation Calculus can be formulated as a logic program. Planning goals can be formulated as prolog goals with a situation variable as in:

```
?- Holds(owns(mary, book1), S), possible(S).
```

Notice that here we use *possible*. Indeed, it is crucial that the planner returns possible plans in which all actions can be executed.

A solution for this query is a situation term S representing a possible situation in which the property is satisfied. S constitutes a *linear plan* because it defines a linear sequence of actions starting from the initial situation s_0 .

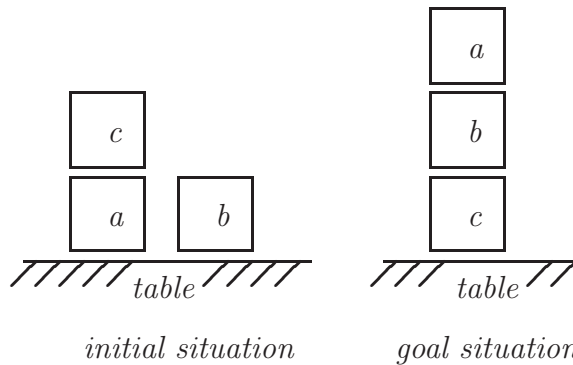
Many prolog systems are unable to solve such planning problems, but there are exceptions (e.g. the A-system [11]). The top-down execution of a planning goal corresponds to a *means-ends analysis*. A means-ends analysis matches the goal of the planning problem with the properties that are initiated by certain actions, adds such an action to the solution plan, and replaces the solved goal with the preconditions of the action. In this way, a planning step transforms subgoals into new subgoals until they are all solved. In general, a subgoal can be solved in different ways, and as a result, a planning step introduces branching in the search space; completeness must be obtained through backtracking.

Exercise 21 *Execute the planning goal*

? – Holds(owns(mary, book1), S), possible(S).

as a logic program. How many solutions are there ? Is the search tree finite or infinite ? Compare your results with using the same theory for prediction studied in the previous section. Try to draw conclusions about complexity for prediction problems versus planning (explanation) problems.

Exercise 22 *Present the following blocks world planning problem in Situation Calculus. Program and execute it in Prolog.*



Exercise 23 *Assignment in imperative programming languages can be regarded as an action which transforms one state of a computer into another state. Consider a program with 3 variables: x , y and z . Initially, they contain the values a , b and c . The problem is to find a sequence of assignments that interchanges the initial values of x and y .*

*Present this problem as a planning problem in Situation Calculus. Program and execute it in Prolog. Notice that this last exercise can be seen as a sort of **automated program development**: the goal represents the postcondition of a procedure, the planner computes a series of actions that constitutes a correct procedure to realise this postcondition.*

3.7 Difficulties of situation calculus

Situation Calculus has shortcomings, both representationally and computationally.

- Dealing with independent sequences of events.
- Events must be totally ordered which makes planning intractable.

3.7.1 Global states and modularity

Situation Calculus employs global states as explicit parameters of time-varying properties. Events are state transformers; every new event induces a new global state. Consider the following scenario:⁵

John and Mary agree to meet in a week. Then they part, and John goes to London while Mary flies to San Francisco. They both lead eventful weeks, each independently of the other, and duly meet as arranged.

Even if we know exactly the order of actions that were carried out by Mary and the order of actions carried out by John, we probably do not know in what order the union of those actions are. In a Situation Calculus term, all of John and Mary's events, although independent, must ordered in a sequence. Since we do not know this order, we cannot represent the outcome of their separated weeks as one situation term. It is difficult to deal with such scenarios in Situation Calculus.

3.7.2 Total ordering of events

Planning in Situation Calculus results in a linear planning system (Section 3.6). The search space of a linear planner consists of all possible (total) orderings of the events in the (intermediate) plans. The number of total orderings grows exponentially with the number of events in the plan which makes linear planning possible only for small problems. In *non-linear planning*, events remain unordered until it becomes necessary to impose a particular ordering. This reduces the search space exponentially.

References

- [6] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence 4*. Edinburgh University Press, 1969.
- [7] Robert A. Kowalski. *Logic for Problem Solving*. Artificial Intelligence Series 7. Elsevier Science Publisher B.V. (North-Holland), 1979.
- [8] Robert A. Kowalski. Data Base Updates in the Event Calculus. *Journal of Logic Programming*, 12(1&2):121–146, 1992.
- [9] Marc Denecker and Eugenia Ternovska. Inductive Situation Calculus. In *Proceedings of Ninth International Conference on Principles of Knowledge Representation and Reasoning, Delta Whistler Resort, Canada*, 2004. URL = http://www.cs.kuleuven.ac.be/cgi-bin/dtai/publ_info.pl?id=41085.

⁵The example is taken from Hayes who proposes the concept of clusters and histories to model action activity in separate regions in space-time. (Hayes, Patrick J., "The second naive physics manifesto", in *Formal Theories of the Commonsense World*, Ablex, ed. Hobbs and Moore, pp. 1–36, 1985)

- [10] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
- [11] Bert Van Nuffelen. *Asystem: An Abductive Constraint System*. URL = <http://www.cs.kuleuven.ac.be/~bertv/Asystem/>.