This paper was selected by a process of
anonymous peer reviewing for presentation at

# COMMONSENSE 2007

8th International Symposium on Logical Formalizations of Commonsense Reasoning

Part of the AAAI Spring Symposium Series, March 26-28 2007,
Stanford University, California

Further information, including follow-up notes for some of the
selected papers, can be found at:

www.ucl.ac.uk/commonsense07

# Deductive Planning with Temporal Constraints

**Martin Magnusson** and **Patrick Doherty**
Department of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
http://www.ucl.ac.uk/commonsense07/
papers/notes/magnusson-and-doherty/

**Abstract**

Temporal Action Logic is a well established logical formalism for reasoning about action and change using an explicit time representation that makes it suitable for applications that involve complex temporal reasoning. We take advantage of constraint satisfaction technology to facilitate such reasoning through temporal constraint networks. Extensions are introduced that make generation of action sequences possible, thus paving the road for interesting applications in deductive planning. The extended formalism is encoded as a logic program that is able to realize a least commitment strategy that generates partial order plans in the context of both qualitative and quantitative temporal constraints.

## 1 Introduction

Temporal Action Logic (TAL) (Doherty &Kvarnström 2006) has proven to be a highly versatile and expressive formalism for reasoning about action and change. One of its distinguishing characteristics is its use of explicit time structures expressed as sub-theories in a standard classical logic. Historically, TAL has used integer time, but it would be interesting from a representational perspective and useful in practical applications to use different temporal structures, or even combinations, such as timepoints and intervals. A practical obstacle in doing this is the inefficiency of a straightforward application of classical theorem proving techniques to deduce inferences from TAL narratives together with axiomatizations of time.

One way to approach this problem is to take advantage of work done with temporal constraint propagation techniques and to combine theorem proving techniques with specialized constraint algorithms for dealing with temporal structures in TAL narratives. In fact, General Temporal Constraint Networks (GTCN) will be used with TAL in order to reason efficiently with a subset of the full formalism. In this manner, it is also possible to combine both time points and time intervals. To do this, we will use a form of semantic attachment where intervals are introduced as terms and relations between intervals as function terms. This makes efficient reasoning with both time points and intervals in TAL possible using GTCN constraint propagation algorithms.

In previous work, we used TAL as a formal specification language for a very powerful forward-chaining planner called TALplanner. The idea was to input a goal narrative formally specified using TAL into the procedural planner which in turn output a plan narrative whose formal semantics was also based on TAL. One could show the formal correctness of the input/output behavior of TALplanner using TAL in this manner.

In contrast, this paper introduces additional extensions to TAL that enable deductive planning at the object level rather than meta-theoretically. In order to do this both action occurrences and sets of action occurrences (a form of plan or narrative) will be introduced as terms and sets in the object language, respectively. In this manner, one may generate partially ordered plans deductively in TAL by appealing to the use of GTCNs together with the TAL deductive mechanism implemented as a logic program.

## 2 Temporal Action Logic

TAL is a narrative based formalism for reasoning about action and change where narratives include action type specifications, causal or dependency constraints, observations at specific points in time, and action occurrences through durations. A high-level macro language is normally used to specify narratives that are compiled into a first-order theory. Circumscription is then used to provide solutions to the frame, qualification and ramification problems. For the purposes of this paper, we will provide narrative examples in first-order logic to describe our extensions and also exclude some details of the full formalism.

In the standard TAL formalism, the predicate Occlude plays a prominent role in the solution to the frame, ramification, qualification and other problems that arise when reasoning about action and change in incomplete contexts. To assert Occlude$(t, f)$ in a theory, provides a constraint that basically states that a fluent $f$ is *allowed* to change value from timepoint $t-1$ to $t$ when using integer time as the basic temporal structure. What value it takes, or if it changes value at all, is determined by additional assertions in the theory such as action occurrences or causal or dependency constraints which are part of TAL narratives. A form of filtered preferential entailment is used where that part of the theory involving the use of the Occlude predicate is first circumscribed, providing a definition of Occlude. This provides a succinct characterization of all timepoint/fluent pairs where change is allowed. Some of the models for this part of the theory may contain spurious change not supported by any

assertions in the theory. These models are filtered by conjoining a persistence formula[1] of the following form:

$$\neg\mathsf{Occlude}(t+1,f) \rightarrow (\mathsf{Holds}(t,f) \leftrightarrow \mathsf{Holds}(t+1,f)) \quad (1)$$

Action occurrences in a TAL narrative are asserted by using an Occurs predicate, where $\mathsf{Occurs}(t_1,t_2,a)$ asserts that an action $a$ occurs during the interval $[t_1, t_2]$. This predicate is also circumscribed, providing a definition of Occurs.

Consider a reasoning domain involving the flight of an Unmanned Aerial Vehicle (UAV) between two locations, expressed in the following TAL narrative, where variables are in *italics* and free variables are assumed universally quantified:

$$\mathsf{Holds}(0,\mathsf{atuav}(\mathsf{base})) \quad (2)$$

$$\mathsf{Occurs}(3,8,\mathsf{fly}(\mathsf{loc1})) \quad (3)$$

$$\mathsf{Occurs}(t_1,t_2,\mathsf{fly}(x)) \rightarrow (\mathsf{Holds}(t_1,\mathsf{atuav}(y)) \rightarrow \mathsf{Holds}(t_2,\mathsf{atuav}(x)) \wedge \neg\mathsf{Holds}(t_2,\mathsf{atuav}(y)) \wedge \forall t,z\,[t_1 < t \leq t_2 \rightarrow \mathsf{Occlude}(t,\mathsf{atuav}(z))]) \quad (4)$$

$$\neg\mathsf{Occlude}(t+1,\mathsf{atuav}(x)) \rightarrow (\mathsf{Holds}(t,\mathsf{atuav}(x)) \leftrightarrow \mathsf{Holds}(t+1,\mathsf{atuav}(x))) \quad (5)$$

The basic modelling primitives are *fluents*, such as atuav in formula 2, that represent properties of the world that might *hold* different truth values over time. Formula 3 is an action occurrence formula. A fluent's truth value is influenced by actions (or causal constraints) as defined by action specification formulas (and dependency constraint formulas). Formula 4 states that the effect of flying the UAV to location $x$ between timepoints $t_1$ and $t_2$ is that at timepoint $t_2$ it will be at location $x$ but no longer be at $y$, provided that it actually started at $y$ at timepoint $t_1$. Note also that the fluent atuav is occluded during the actions duration.

Circumscribing Occurs in the narrative results in a definition characterized by formula 6. Circumscribing Occlude in the action specification formula subset of the narrative (formula 4) results in a definition characterized by formula 7.

$$\mathsf{Occurs}(t_1,t_2,f) \leftrightarrow t_1 = 3 \wedge t_2 = 8 \wedge f = \mathsf{fly}(\mathsf{loc1}) \quad (6)$$

$$\exists t_1,t_2,y,z\,[t_1 < t \leq t_2 \wedge \mathsf{Occurs}(t_1,t_2,\mathsf{fly}(z)) \wedge \mathsf{Holds}(t_1,\mathsf{atuav}(y))] \leftrightarrow \mathsf{Occlude}(t,\mathsf{atuav}(x)) \quad (7)$$

Formula 5 is a persistence formula for atuav which together with formula 7 may be used to prove persistence and non-persistence of the fluent atuav on the time-line. Given the example narrative we can prove that the UAV will be at loc1 at timepoint 8 by proving that it remains at base until the fly action takes it to loc1 between timepoints 3 and 8.

This short introduction to TAL serves as a basis for understanding the extensions described in this paper but is by no means a complete feature list. Some topics that have been dealt with in other publications are e.g. ramifications, qualifications, context-dependent effects, side effects, and non-deterministic actions. For a more detailed presentation of TAL the reader is referred to (Doherty &Kvarnström 2006).

[1] Actually, in the current version of TAL, a number of persistence formulas are used, one per fluent specified as being persistent. Moreover, the Holds predicate is usually generalized to include non-boolean fluents using a third argument from the fluents' value domain. For the purposes of this paper, these details are not important.

## 3 Reified Action Occurrences

TAL action occurrences are specified using the Occurs *predicate*. Hypothesizing new instances of this predicate given a goal would require abductive techniques or the use of some special-purpose planning algorithm such as TALplanner. In order to experiment with *deductive* planning techniques using TAL, it is necessary to reify action occurrences as terms in the language so one can reason with and quantify over actions.

We introduce a function occ and replace action occurrences $\mathsf{Occurs}(t_1,t_2,a)$ by terms of the form $\mathsf{occ}(t_1,t_2,a)$. A set of action occurrences is a collection of such terms and, since their order in the collection is unrelated to their actual temporal order determined by their relations to the explicit time line, the collection behaves like a regular mathematical set. Additionally, a vital property is incompleteness. A reasoning problem involving a fully specified set of action occurrences often corresponds to a prediction or postdiction problem while an under-specification often gives rise to a planning problem.

We would like to add individual action occurrence terms to action occurrence sets. We adopt the (infix) fluent composition function $\circ$, used to represent world states in the Fluent Calculus as described in e.g. (Thielscher 2005), to represent TAL action occurrence sets as terms. So, if $a$ is an action occurrence term and $p$ is an action occurrence set, then $a \circ p$, although a term, essentially represents the set of action occurrences $\{a\} \cup p$. E.g. a set $p$ involving a fly action and possibly other actions can be written as:

$$\exists p'\,[p = \mathsf{occ}(3,8,\mathsf{fly}(\mathsf{loc1})) \circ p']$$

Thielscher provides axioms that characterize the behaviour of $\circ$ in (2005), but we will be content with introducing a new Occurs predicate, the meaning of which is understood through semantic attachment as follows:

$$\mathsf{Occurs}(t_1,t_2,a,p) \stackrel{\mathsf{def}}{=} \mathsf{occ}(t_1,t_2,a) \in p,\ p \text{ is formed using } \circ$$

The old Occurs predicate had direct logical consequences through action specifications such as formula 4 above. We can deduce the same logical consequences from reified occurrences by adding an action set argument to the Holds and Occlude predicates. E.g. formula 4 is rewritten as:

$$\mathsf{Occurs}(t_1,t_2,\mathsf{fly}(x),p) \rightarrow (\mathsf{Holds}(t_1,\mathsf{atuav}(y),p) \rightarrow \mathsf{Holds}(t_2,\mathsf{atuav}(x),p) \wedge \neg\mathsf{Holds}(t_2,\mathsf{atuav}(y),p) \wedge \forall t,z\,[t_1 < t \leq t_2 \rightarrow \mathsf{Occlude}(t,\mathsf{atuav}(z),p)])$$

When using persistency of fluents it becomes necessary to prove non-occurrence of actions with effects that might otherwise modify the fluent's value. The TAL circumscription policy usually makes this possible by circumscribing the *Occurs* predicate. With reified action occurrences, assumptions made about the plan argument $p$ can be expressed as formulas. E.g. the formula $\neg\exists t_1,t_2,l\ [\mathsf{Occurs}(t_1,t_2,\mathsf{fly}(l),p)]$ excludes *fly* actions from a specific action occurrence set $p$. Such assumptions can be proven to hold when any unbound plan variables have been instantiated. However, the logic programming methodology presented in Section 6 uses a set of constraint handling rules to continually track and resolve potential conflicts between action effects and persistency.

## 4 Interval Occlusion

The TAL occlusion concept provides a means to control at which timepoints fluents may change value. But in our experience, occlusion *between* specific timepoints is often just as usueful as occlusion *at* specific timepoints. In fact, one may think of negated occlusion (persistence) of a fluent in terms of intervals. Introducing intervals as first-class citizens in TAL makes interval-based temporal constraint formalisms applicable for temporal reasoning, and they simultaneously provide a meaning for the intervals through semantic attachment, relating interval and timepoint primitives.

As a first step in this direction we introduce the predicate Occlude($t_1$,$t_2$,$f$) to assert what we will call *interval occlusion* for a fluent $f$. The intended meaning is that fluent $f$ is interval occluded over $(t_1, t_2]$ if it is occluded at some timepoint in that interval. Conversely, if we manage to prove that fluent $f$ is *not* interval occluded for $t_1$ and $t_2$, then we know that its value will persist. Formally, we define interval occlusion in terms of the regular timepoint occlusion as:

$$\text{Occlude}(t_1,t_2,f) \leftrightarrow \exists t\ [t_1 < t \leq t_2 \wedge \text{Occlude}(t,f)] \quad (8)$$

It is not difficult to construct a proof through induction over the length of intervals that (8) together with the persistence formula for timepoint occlusion (1) entails an *interval persistence* formula:

$$\neg\text{Occlude}(t_1,t_2,f) \rightarrow \forall t\ [t_1 < t \leq t_2 \rightarrow (\text{Holds}(t-1,f) \leftrightarrow \text{Holds}(t,f))] \quad (9)$$

Note that even if a fluent is interval occluded over a given interval, it might still be unoccluded in sub-intervals. However, if the fluent is interval persistent over the interval it must also be persistent in all sub-intervals.

Formulas 8 and 9 entail the following formula:

$$\neg\text{Occlude}(t_1,t_2,f) \rightarrow (\text{Holds}(t_1,f) \leftrightarrow \text{Holds}(t_2,f)) \quad (10)$$

Using (10), the truth value of a fluent can be made to "jump" any number of timepoints in a single proof step. This technique is essential in the implementation that is described in section 6. Though not much has been gained if we still need to prove timepoint occlusion false at each individual timepoint before being able to prove interval occlusion false using its definition formula 8. Instead, we would like to introduce interval occlusion directly in the occlusion formula 7 of our UAV example, repeated here for convenience:

$$\exists t_1,t_2,y,z\ [t_1 < t \leq t_2 \wedge \text{Occurs}(t_1,t_2,\text{fly}(z)) \wedge \text{Holds}(t_1,\text{atuav}(y))] \leftrightarrow \text{Occlude}(t,\text{atuav}(x)) \quad (11)$$

By substituting the timepoint occlusion predicate in the definition of interval occlusion in formula 8 by its definition in (11), with timepoint variables renamed and existential quantifiers rearranged, we get:

$$\exists t_3,t_4,y,z\ [\exists t\ [t_1 < t \leq t_2 \wedge t_3 < t \leq t_4] \wedge \text{Occurs}(t_3,t_4,\text{fly}(z)) \wedge \text{Holds}(t_3,\text{atuav}(y))] \leftrightarrow \text{Occlude}(t_1,t_2,\text{atuav}(x))$$

The sub-formula $\exists t\ [t_1 < t \leq t_2 \wedge t_3 < t \leq t_4]$ asserts the existence of a timepoint that is common to the intervals $(t_1, t_2]$ and $(t_3, t_4]$. Hence, the fluent expressing the location of the UAV is interval occluded in $(t_1, t_2]$ iff a fly action, which is the only action affecting the UAV's location in our example, occurs in an interval that *overlaps* with $(t_1, t_2]$ in any way, given that the precondition of the fly action is satisfied.

## 5 Temporal Constraint Networks

The second step towards introducing intervals as first-class citizens in TAL is the application of an interval-based temporal constraint formalism. By adopting the general temporal constraint networks, developed by Meiri (1996), we gain a temporal formalism that is complete for a large class of temporal reasoning problems. Generalizing Allen's interval algebra (1983), Vilain and Kautz' point algebra (1986), and formalisms based on metric constraints, the GTCN facilitates both qualitative and quantitative reasoning with incomplete information and includes both interval and timepoint primitives that provide a natural fit with our requirements.

We replace timepoint pairs by intervals in the Occurs($t_1$,$t_2$,$a$) and Occlude($t_1$,$t_2$,$f$) predicates to obtain an Occurs($i$,$a$) and an Occlude($i$,$f$) predicate. The Holds predicate still accepts timepoint arguments and these timepoints are related to intervals by *Point-Interval* (PI) relations, timepoints are related to each other by *Point-Point* (PP) relations, and intervals to each other by *Interval-Interval* (II) relations. When action occurrences are temporally ordered by the PI, PP, and II relations, instead of relying on integers and their inherent ordering, we need to represent those relations as a fundamental part of the action occurrences. To this end we introduce pi($t$,$i$,$r$), pp($t_1$,$t_2$,$r$), and ii($i_1$,$i_2$,$r$) functions that are used to represent the possible relations $r$ given in (Meiri 1996). These functions are incorporated into the formalism by extending action occurrence sets to include both action occurrence terms and interval relation terms. For example, the following action occurrence set represents a fly action that takes place during interval i1, which starts at timepoint tp1 and ends at timepoint tp2.

occ(i1,fly(loc1)) ∘ pi(tp1,i1,[starts]) ∘ pi(tp2,i1,[finishes])

To make use of this representation we introduce new predicates where the GTCN constraints expressed in the $p$ argument on the left hand side encodes the PI and II relations on the right hand side:

$$\text{Starts}(t,i,p) \stackrel{\text{def}}{=} t\{s\}i$$
$$\text{Finishes}(t,i,p) \stackrel{\text{def}}{=} t\{f\}i$$
$$\text{Overlaps}(i_1,i_2,p) \stackrel{\text{def}}{=} i_1\{o,s,d,f,=,fi,di,si,oi\}i_2$$

The Overlap predicate, in effect, implements the interval occlusion overlap sub-formula $\exists t\ [t_1 < t \leq t_2 \wedge t_3 < t \leq t_4]$ using the constraint network.

## 6 Planning as Deduction

Consider a logistics scenario, extending the UAV example from Section 2. The task is to deliver crate1 and crate2 from home base to another location loc1. The UAV needs help from an unmanned ground vehicle (UGV) that attaches the crates to the UAV while it is hovering to remain absolutely

still. The TAL enhancements introduced above pave the way for deductive planning in scenarios similar to this.

We introduce a Prolog implementation of a reasoning system for a subset of TAL, called PARADOCS, that views Planning And Reasoning As DeductiOn with ConstraintS. It supports prediction from a fully instantiated set of actions, planning from the empty set of actions, and anything in between. PARADOCS can be used to reason about a subset of TAL narratives that can be encoded using Horn clauses that are in close correspondence with the first-order TAL axioms. Temporal relations are managed by a general temporal constraint network implemented by a set of constraint handling rules (CHR) (Frühwirth 1994) (Frühwirth 1998). Encoding the above scenario using the PARADOCS framework results in a quite compact program. The entire implementation, excluding the code for the GTCN solver, is presented and explained below.

We start with the semantic attachments. The composition function comp($a$,$p$,$p'$), borrowed from Thielscher's FLUX implementation (Thielscher 2005), denotes the composition $p = a \circ p'$. The other predicates use the GTCN to link intervals with their start and end timepoints, to give intervals duration, and to relate intervals with each other.

```
occurs(I,A,P) :-
  comp(occ(I,A),P,Pp), circ_occurs(I,A).
comp(A,[A|Pp],Pp).
comp(A,P,[A1|Pp]) :-
  nonvar(P), P = [A1|P1], A \== A1,
  comp(A,P1,Pp).
duration(I,T1,T2,Tmin,Tmax) :-
  arc(T1,I,[starts],p-i),
  arc(T2,I,[finishes],p-i),
  arc(T1,T2,[Tmin-Tmax],p-p).
not_overlap(I,I2) :-
  arc(I,I2,[after,before,meets,met_by],i-i).
covers(I,I2) :-
  arc(I,I2,[contains,equals,
            finished_by,started_by],i-i).
```

Next we deal with persistence. While planning, one is continually using non-occlusion to prove fluent value persistence while, at the same time, actions are added to the reified action occurrence set that threaten to affect and modify the very same occlusion. Consequently, we need to continually ensure that considered actions do not violate any persistence proofs that have already been assumed. The CHR framework together with the temporal constraint network make a flexible solution to this problem possible.

Whenever a non-occlusion assumption is used to prove the persistence of a fluent value over an interval, this is noted using the `not_occlude` constraint. Additionally, whenever an action is added to the plan, a `circ_occurs` constraint is added, which serves as a representation of the circumscription of action occurrences added so far. The definition of occlusion, obtained through circumscription of the *Occlude* predicate in the original narrative, is compiled into a set of CHR rules. These trigger on potential non-occlusion/action-occurrence conflicts, enforcing new temporal constraints in the network that resolve the conflict by making sure the action interval does not overlap any part of the persistence interval. If such a resolution is not possible the network becomes inconsistent, triggering a backtrack in the planning process that will consider other actions or other ways of proving the fluent value. By expressing occlusion using constraint handling rules we ensure that assumptions are automatically triggered and reevaluated whenever needed.

In our case every fluent is persistent except `still`, a *durational* fluent that is true while the UAV is hovering and reverts to false as soon as it stops hovering. Formula 10 is put in Horn form to express persistence in the forward direction. We exclude `still`, add a `persist` constraint (explained in the next paragraph), and abridge the temporal relations using `duration`, to get the persistence formula and conflict resolution rules:

```
holds(T2,F,P) :-
  F \== still, persist(I,F),
  inf(Inf), duration(I,T1,T2,0,Inf),
  holds(T1,F,P), not_occlude(I,F).

not_occlude(I,atuav(Y)), circ_occurs(I2,fly(X))
  ==> not_overlap(I,I2).
not_occlude(I,at(X,Y)), circ_occurs(I2,attach(X))
  ==> not_overlap(I,I2).
not_occlude(I,at(X,Y)), circ_occurs(I2,drop(X))
  ==> not_overlap(I,I2).
```

There are potential problems with infinite looping given the depth-first search strategy of Prolog. This is especially true concerning the persistence formula. A fluent is true at a timepoint if it is true at an earlier timepoint and is not occluded during the interval in between. It is true at the earlier timepoint if it is true at an even earlier timepoint, and so on. We see that the looping is correct but unwanted behaviour. But the observation that if a fluent is persistent over two intervals that meet, then it must be persistent over the union of the intervals, provides the key to a simple solution. To gain completeness in the general case we would be forced to add a plan length or search depth limit, but here we get by with adding a constraint rule that prohibits two consecutive persistence intervals:

```
persist(I1,F), persist(I2,F),
  path(_,I2,I1,[met_by],i-i,_) ==> fail.
```

The initial state can be specified after resolving one complication. The GTCN works exclusively with variables so we can not use a dedicated constant to denote a first timepoint. Instead we use a constraint `now` that makes sure all references to the initial timepoint are equal. Recognizing that the initial state holds regardless of what actions are in the action set argument we get:

```
now(T1) \ now(T2) <=> T1 = T2.
holds(Now,atuav(base),P) :- now(Now).
holds(Now,atugv(base),P) :- now(Now).
holds(Now,at(crate1,base),P) :- now(Now).
holds(Now,at(crate2,base),P) :- now(Now).
```

Moving on to the action specification formulas one might wonder how negative action effects are realized without regular negation. The answer is that the conflict resolution constraints, introduced previously, prevent the application of fluent persistency over an interval where an action with a negative effect for that fluent value overlaps. E.g. if a narrative contains an action that flies the UAV from the base to another location, we can no longer use persistence to prove the UAV is still at the base at some timepoint after or during

the flight. The only way of proving that the UAV is at the base would be by flying it back.

We encode the action formulas to `fly` the UAV, to use the UGV to `attach` a crate to the UAV while it executes a `hover`, and to `drop` a crate at the current location, adding two constraints on the concurrency of flying and hovering:

```
holds(T2,atuav(X),P) :-
  occurs(I,fly(X),P), holds(T1,atuav(Y),P),
  traveltime(X,Y,T), duration(I,T1,T2,T,T).
holds(T2,carrying(X),P) :-
  occurs(I,attach(X),P),
  holds(T1,at(X,Y),P), holds(T1,atugv(Y),P),
  holds(T1,atuav(Y),P), holds(I2,still,P),
  covers(I2,I), duration(I,T1,T2,60,60).
holds(I,still,P) :-
  occurs(I,hover,P),
  inf(Inf), duration(I,T1,T2,0,Inf).
holds(T2,at(X,Y),P) :-
  occurs(I,drop(X),P), holds(T1,carrying(X),P),
  holds(T1,atuav(Y),P), not_occlude(I,atuav(Y)),
  duration(I,T1,T2,10,10).

circ_occurs(I,hover), circ_occurs(I2,fly(X))
   ==> not_overlap(I,I2).
circ_occurs(I,fly(X)), circ_occurs(I,fly(Y))
   ==> X = Y.
```

Note that the precondition of `attach` requires the fluent `still` to hold at all timepoints during the operation. In first-order logic this is solved with a universal quantifier, but Prolog is less expressive. Instead we take advantage of the fact that all timepoints between two timepoints, taken together, constitute an interval. The `hover` action thus causes the `still` fluent to be true over the interval during which the hover is active.

One of the distinguishing qualities of deductive planning is the potential for a seamless use of background knowledge. Knowledge about actions and knowledge about other things are expressed in a unified declarative format that is operated upon using the same deductive mechanisms. Although this description might not be a complete fit with present-day deductive planning systems, it still provides a strong and appealing intuition. As a tiny gesture towards this possibility we provide the system with some "background knowledge" expressing the duration of travel given the speed of the UAV and the distance between locations:

```
speed(25).
coord(base,20000,12000).
coord(loc1,26000,20000).
coord(loc2,26000,12000).
dist(A,B,D) :-
  coord(A,Ax,Ay), coord(B,Bx,By),
  D is sqrt(exp(Bx - Ax,2) + exp(By - Ay,2)).
traveltime(X,Y,T) :-
  dist(X,Y,D), speed(S),T is (D / S) * 1.5.
```

Finally, for completeness of this presentation, we include the last three constraint rules that remove redundant constraints:

```
not_occlude(I,F) \ not_occlude(I,F) <=> true.
circ_occurs(I,A) \ circ_occurs(I,A) <=> true.
persist(I,F) \ persist(I,F) <=> true.
```

### 6.1 Plans are Temporal Networks

Using the encoding we can solve the UAV scenario by letting Prolog prove a goal such as the following:

```
?- now(Now), P = [occ(Id,drop(crate1))|P2],
   arc(Td,Id,[finishes],p-i),
   arc(Now,Td,[600-900],p-p),
   holds(Tn,at(crate1,loc1),P),
   holds(Tn,at(crate2,loc1),P).
```

In addition to the declarative goals that the crates should be at `loc1` we state that the resulting plan will involve dropping `crate1` and that this action should be completed within 10 to 15 minutes from now.

After a pause of 17 seconds SICStus Prolog 3.12.5, running the above program on an Intel Pentium M 1.8 GHz, produces a plan in the form of a set of actions and eight pages of constraints that relate them temporally. The plan is visualized using a graphical interface that displays a diagram approximately like the one in Figure 1.

The diagram is a convenient overview of the plan but does not necessarily reflect the full extent of uncertainty in the corresponding GTCN. Through interaction with the graphical interface we discover that the two `attach` actions, as well as the two `drop` actions, can be ordered in whatever way seems best, that the `hover` action is constrained only to be before the `fly` action while covering the `attach` actions, and that there is flexible room for pauses between actions during execution. The quantitative constraint on `crate1` has been propagated and shows e.g. that the UAV needs to leave base at 4:50 at the latest, and that the drop can be made at 11:10 at the earliest.

Let us now suppose that the plan is executed, the UGV attaches the crates and the UAV flies to `loc1`. However, when it arrives a UAV operator pauses the execution and views the remaining parts of the plan, the two drop actions that have not yet been executed. The operator changes the goal of delivering `crate2` at `loc1` to another location, `loc2`.

```
?- now(Now), P = [occ(Id,drop(crate1)),
                  occ(Id2,drop(crate2))|P2],
   holds(Tn,at(crate1,loc1),P),
   holds(Tn,at(crate2,loc2),P).
```

The persistence intervals ensure a large amount of flexibility in the plan. PARADOCS takes advantage of this and elaborates the plan fragment to fit the goals by introducing a new `fly` action in between the `drop` actions. New actions can be inserted in the middle of the plan without first backtracking on later actions, thereby reducing the search space. In fact, the UAV problem just discussed was solved without backtracking on any action choices.

## 7 Related Work

Much of the inspiration to PARADOCS comes from other research groups with similar approaches. We only have space to compare our approach with the most important influences and start with Shanahan's abductive event calculus planner (2000). It is based on the Event Calculus, another explicit time formalism, but produces its partially ordered plans using abduction on Occurs and temporal relations through an abductive meta-interpreter, which is (we think) slightly more complex than our deduction with reified action occurrences.

Unlike Shanahan's use of timepoint relations the expressive power of the GTCN does not force us to adopt the

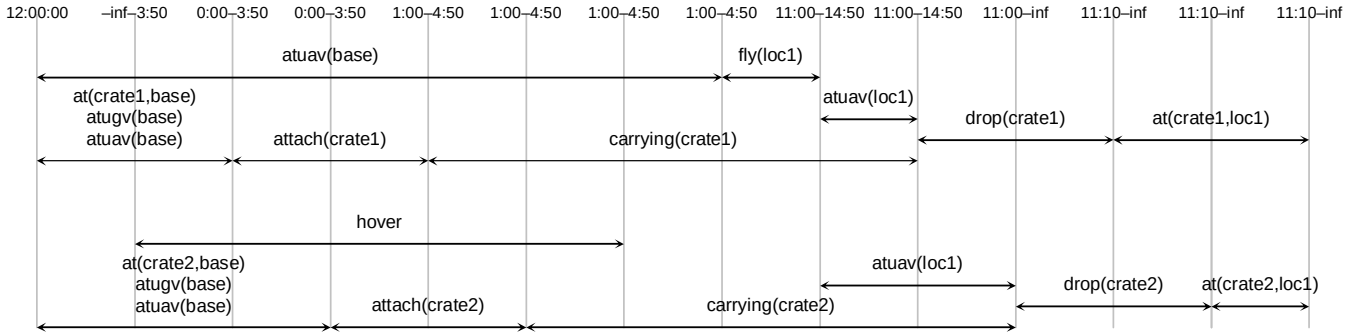


Figure 1: A plan diagram.

promote/demote strategy from partial order planning algorithms when faced with possible "protected link" threats. Instead, persistence/action conflicts can be detected even with incomplete temporal information. We have in fact experimented with weaker constraint solvers similar to simple temporal networks, but found the much added complexity of the implementation that results from the introduction of promotion and demotion to detract from the clarity of presentation of the basic PARADOCS mind set.

In a feature comparison, the event calculus planner extends our basic planning capabilities with hierarchical planning and a form of knowledge producing sensing actions based on abduction (Shanahan &Witkowski 2001).

Another *deductive* planning framework is Golog (Levesque *et al.* 1997), based on the Situation Calculus from where the idea of reified action occurrences passed around using an extra predicate argument originates. But Golog's situation terms contain linearly ordered action sequences without explicit temporal information. This fact prevents the generation of partially ordered plans, but note that such shortcomings can be overcome through various extensions, as is done e.g. in Congolog (Giacomo, Lesperance, &Levesque 2000).

Finally, the Fluent Calculus serves as the formal basis for FLUX (Thielscher 2005), another logic programming methodology that supports deductive planning with linear plans. Constraint handling rules are used in FLUX, but not for temporal reasoning. Instead, they enable the representation and planning with some forms of incomplete information.

## 8 Conclusions

The explicit time formalism of Temporal Action Logic exposes qualitative and quantitative temporal primitives that are particularly amenable to reasoning using temporal constraint networks. This paper takes the first steps in that direction through extensions to TAL and a concise but expressive logic program for deductive planning. The concept of occlusion and the use of general temporal constraint networks enable a novel solution to persistency threats where a set of constraint handling rules implements a strategy of minimal commitment by taking advantage of disjunctive and incompletely specified temporal constraints. Furthermore, non-occlusion constraints in solution plans represent fluent value dependencies that can be monitored during execution, and partially executed plans with failed dependencies can be reasoned with and completed in the same deductive framework.

## Acknowledgements

This work is funded in part by a grant from the Swedish Research Council (VR) and from the Swedish National Aeronautics Research Program (NFFP).

## References

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

Doherty, P., and Kvarnström, J. 2006. *The Handbook of Knowledge Representation*. Elsevier. chapter 18. To appear.

Frühwirth, T. 1994. Temporal reasoning with constraint handling rules. Technical Report ECRC-94-05, Munich, Germany.

Frühwirth, T. 1998. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming* 37(1-3):95–138.

Giacomo, G. D.; Lesperance, Y.; and Levesque, H. J. 2000. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1-2):109–169.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1-3):59–83.

Meiri, I. 1996. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence* 87(1–2):343–385.

Shanahan, M., and Witkowski, M. 2001. High-level robot control through logic. *Lecture Notes in Computer Science* 1986.

Shanahan, M. 2000. An abductive event calculus planner. *Journal of Logic Programming* 44(1-3):207–240.

Thielscher, M. 2005. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming* 5(4–5):533–565.

Vilain, M., and Kautz, H. 1986. Constraint propagation algorithms for temporal reasoning. In *Proceedings of AAAI-86*, 377–382.