

Planning through the TRLi temporal reasoning system

C.C. Marinagi^a, T. Panayiotopoulos^b, C.D. Spyropoulos^a

*^aInstitute of Informatics and Telecommunications,
N.C.S.R.Demokritos, 153 10 Aghia Paraskevi, Athens, Greece*

*^bDepartment of Informatics, University of Piraeus, 80 Karaoli &
Dimitriou Str., 185 34 Piraeus, Greece*

Abstract

In this paper we propose a planning system based on the TRLi temporal reasoning system. The planning algorithm incorporates TRLi as a temporal deduction component with temporal constraint solving capabilities. The innovations of the planning system are the enhanced expressive power, the flexibility and the reusability of the generated plans, as well as the simplification of the action ordering mechanism as it orders temporal references instead of restructuring a whole plan network.

1 Introduction

Artificial Intelligence work in planning has provided us with planning methods, formalisms and systems which have been applied in many areas such as process control, robotics, scheduling, manufacturing logistics, mission logistics, project management, risk management, production control, etc.

During the last decade, there has been much research activity concerning the problem of incorporating time into planning. Deviser [11] has been one of the first attempts to incorporate time as a key component of planning. More recent approaches build a non-linear planner on top of a time map management system [4,5,10]. Allen's approach expresses the planning problem as a temporal logic's set of axioms [1], while Cheesman's approach integrates a set of axioms which implement some kind of temporal reasoning into the planning algorithm [2].

The TRLi system [7,8,9] has been recently proposed as an alternative approach to temporal reasoning. Its operation is based on temporal constraint solving techniques which are implemented by a symbolic constraint solver. TRLi has enhanced representation capabilities of temporal references and its syntax is

20 Artificial Intelligence in Engineering

very similar to that of Prolog. It has already been used in various domains such as intelligent problem solving, simulation and temporal databases [9].

In this paper we propose a planning algorithm based on the TRLi temporal reasoning system. The planning algorithm incorporates calls to both the TRLi interpreter and the symbolic constraint solver of TRLi. In the following section we present the TRLi system very briefly, and in the next two sections we describe the action representation schema and the planning algorithm. Finally we make some comparisons with existing systems and provide some ideas for future work.

2 The TRLi temporal reasoning system

TRLi has been developed as a meta-level interpreter (TRLi interpreter) which in collaboration with a symbolic constraint solver (SCS solver), handles the temporal world and the queries as input data, and provides the answers as output data. TRLi expresses temporal information as *temporal references* to predicates such as the *temporal point* T , the *temporal interval* $\langle T1, T2 \rangle$, and the *temporal instance* $[T1, T2]$.

Predicates referenced in such a way are called *extended predicates* or *extended atoms*. There are three categories of predicates : classical predicates, extended predicates and temporal predicates. *Classical predicates* bear no temporal reference on them, i.e. these are predicates normally found in Prolog. Extended predicates are further subcategorised into *properties* and *events*. *Temporal predicates* must be interpreted as relations among their temporal arguments.

For classical atoms, we retain SLD-resolution, i.e. TRLi behaves like Prolog. For extended atoms, however, we have defined additional inference rules. These rules have been exactly defined in [8].

In order to utilise TRLi for planning we have only needed property atom types. The inference rules for property atom types are :

- [A1] $\langle u1, u2 \rangle : p \vdash \langle l1, l2 \rangle : p$, where $\{u1, \dots, u2\}$ contains $\{l1, \dots, l2\}$.
- [A2] $\langle u1, u2 \rangle : p \vdash [l1, l2] : p$, where $\{l1, \dots, l2\}$ overlaps with $\{u1, \dots, u2\}$.
- [A3] $[u1, u2] : p \vdash [l1, l2] : p$, where $\{l1, \dots, l2\}$ contains $\{u1, \dots, u2\}$.
- [A4] $\{\langle u1, u2 \rangle : p, \langle u3, u4 \rangle : p\} \vdash \langle u0, u5 \rangle : p$, where $\{u1, u2\}$ is overlapping with $\{u3, u4\}$, and when concatenated they produce $\{u0, \dots, u5\}$.

In the following lines we will briefly present some examples of these rules. From $\langle 8, 12 \rangle : \text{on}(a, b)$ infer $\langle 9, 11 \rangle : \text{on}(a, b)$, i.e. if block a was on block b from 8 to 12, then block a was on block b from 9 to 11. From $\langle 8, 10 \rangle : \text{ontable}(c)$ infer $[7, 9] : \text{ontable}(c)$, i.e. if block c was on the table from 8 to 10, then block c was also on the table at some time between 7 and 9. From $[8, 9] : \text{clear}(a)$ infer $[7, 10] : \text{clear}(a)$, i.e. if block a was clear at some time between 8 to 9, then block a was also clear at some time between 7 to 10. From $\langle 5, 10 \rangle : \text{holding}(b)$ and $\langle 7, 12 \rangle : \text{holding}(b)$, infer $\langle 5, 12 \rangle : \text{holding}(b)$, i.e. if the robot was holding block b from 5 to 10, and from 7 to 12 then the robot was holding block b from 5 to 12.

3 Action representation

In the proposed formalism we use an action representation schema (Figure 1) where actions, conditions and effects are temporally qualified.

We adopt two types of conditions, called *assumption* and *subgoal* conditions which correspond to 'only-use-if' and 'achieve' conditions of O-Plan [3], respectively. Both assumptions and subgoals are checked to be true, while subgoals can also be expanded by selecting actions.

The effects are distinguished in *primary-effects* and *general-effects*. Primary are those effects that provide a reason for the particular action to be called, while general effects, are those that are produced as side effects when the action is executed [3].

Actions, conditions and effects are represented as TRLi extended atoms, i.e. they are classical predicates labelled by temporal references (temporal points, temporal intervals, temporal instances). The duration of each action occurrence (or execution) is also determined with a temporal constant.

Using the temporal references a wide range of cases can be represented : *instant actions*, which occur at a point of time (temporal point), *interval actions*, which occur over an interval (temporal interval), *actions with uncertain occurrence* which are known to occur at some time between two time points

action= $\langle N, D, Cd, E, Cst \rangle$, where

N is the action's parameterized name,

D is the duration,

Cd is the conditions list, $Cd = A \cup S$,

where A is the assumptions list and S is the subgoals list,

E is the effects list, $E = PE \cup GE$,

where PE is the primary effects list, GE is the general effects list,

Cst is the temporal constraints list.

Figure 1. Action representation schema

$action(\langle T1, T2 \rangle : pickup(X)), 10,$ $[[\langle T3, T4 \rangle : ontable(X)]],$ $[\langle T5, T6 \rangle : clear(X)]],$ $[[\langle T1+9, T8 \rangle : holding(X)],$ $[],$ $[T3 < T1, T5 < T1, T4 \geq T1, T6 \geq T1]]$	$action(\langle T1, T2 \rangle : unstack(X, Y)), 20,$ $[[\langle T3, T4 \rangle : on(X, Y)]],$ $[\langle T5, T1+3 \rangle : clear(X)]],$ $[[\langle T7, T8 \rangle : holding(X)], \langle T2, T10 \rangle : clear(Y)],$ $[],$ $[T3 < T1, T5 < T1, T4 \geq T1, T7 > T1, T7 \leq T2])$	N,D A S PE GE Cst
$action(\langle T1, T2 \rangle : stack(X, Y)), 20,$ $[],$ $[\langle T3, T4 \rangle : clear(Y), \langle T5, T1+12 \rangle : holding(X)]],$ $[[\langle T7, T8 \rangle : on(X, Y),$ $[\langle T9, T10 \rangle : clear(X)]],$ $[T3 < T1, T5 < T1, T4 \geq T1, T7 > T1, T7 \leq T2, T9 > T1, T9 \leq T2])$		N,D A S PE GE Cst

Figure 2. Description of action schemata for the blocks world example

22 Artificial Intelligence in Engineering

(temporal instance). Similar cases for conditions and effects can also be represented. Temporal references contain either *absolute temporal points*, i.e. specific time points, or *relative temporal points*, i.e. temporal expressions of the form $T \pm n$, where T is a temporal variable appearing at some effect and/or condition of the action, and n is an integer that represents n time points (temporal duration) after/before T .

The last element of the action representation schema is the *temporal constraint list* which contains temporal relations between time points appearing as variables in the schema. In this way, the user has the capability to specify any kind of reasonable temporal constraints. Thus, the user, in addition to normal conditions and effects, may specify *retroactive conditions* that end before the start of the action, *postactive conditions* that hold during a part of the action, as well as *delayed effects* that start after the end of the action. As an example consider the definition of a set of actions from the blocks world in Figure 2. The proposed action representation schema is an extension of the classical STRIPS-like formalisms [6] as it provides enhanced representation capabilities.

4 The Planning Algorithm

The proposed planning algorithm follows the basic steps of the classical planning algorithms, but it produces the final plan by calling the TRLi interpreter and the TRLi symbolic constraint solver. Therefore, the generated plan is guided by the temporal inferences and constraint solving techniques of the TRLi system. During plan construction the following lists are used :

- The *World Description list* (WD list). At the beginning of plan construction it contains the initial world description, that is a set of properties that hold over particular time intervals. As planning proceeds the WD list is updated with the action's effects each time a new action is chosen. A WD entry has the form of a TRLi clause, which is either a *temporal fact* of the form $T_i:p_i$, or a *temporal rule* of the form $T_0:p_0 :- T_1:p_1, \dots, T_n:p_n$, where T_i is a temporal reference and p_i is a classical predicate.

- The *current Plan list* (P list), which contains the actions that have been chosen up to the particular point of the search space. A P entry has the form $T:action$, where T is the temporal reference of the action.

- The *temporal Constraints list* (C list), which contains all constraints that are valid up to the particular point. When a new constraint is attempted to be added it is first checked for consistency by the SCS symbolic constraint solver. A C entry has the form $T1 \text{ rel } T2$, where $\text{rel} \in \{<, >, =, \leq, \geq\}$.

These lists give the Status of the current plan as it has been developed up to this point. So, at each particular step of the algorithm a *Status_in*= $\langle WD_{in}, P_{in}, C_{in} \rangle$ and a *Status_out*= $\langle WD_{out}, P_{out}, C_{out} \rangle$ is maintained. The change of plan Status is not a state change as in STRIPS [6], because the plan Status does not change only when an action is selected (or an operator is applied), but also when either the WD list or the C list have been changed. Moreover, Status

gives global information of the plan up to the current point, it is not just a snapshot of the world.

There are also three other lists that are used during plan construction:

- The *Unsatisfied Goals list* (UG list). At the beginning of plan construction the UG list contains the 'top goals' that must be satisfied. As planning proceeds this list is updated. Those goals that become satisfied are extracted and new subgoals are inserted that come up when an action is selected.
- The *Non Ground Effects list* (NGrE list). It contains all effects with unbound variables. Instead of inserting them into the world description, these are kept in NGrE list until they take a value. When they take a specific value, they are moved to GrE list.
- The *Ground Effects list* (GrE list). It contains effects with bound variables. When Status consistency is maintained, these ground effects are moved to the WD list.

In our formalism we have also adopted the '*persistence assumption*' [1.4], i.e. a World Description entry remains true until explicitly changed. Therefore, instead of coding any world changes as add-delete lists, we use lists of contradicted predicates. These *contradicted_lists* determine a predicate and a list of predicates, so that each of the later cannot be simultaneously true with the former.

The plan is generated backwards, starting from goals. The plan algorithm is described in the following lines in a verbal style, while in figure 3 a Prolog-like description of the algorithm is given.

TRLPlanner :

While UG list $\neq \{\}$ do

Extract the first unsatisfied goal G from UG list, try to satisfy G, extract G from UG list and call recursively TRLPlanner with the new Status.

Satisfy Goal :

Use TRLi to examine if G is already true in <WDin,Cin>. Insert ground effects.

If G can not be proved in <WDin,Cin>, select an action to achieve G, such that one of its primary effects matches with G. During action selection the SCS solver is used to update the Cin list. Keep all non ground effects of the selected action in the NGrE list. Expand the action.

Expand the action :

Use TRLi to examine if each assumption of the selected action is true in <WDin,Cin>. Insert ground effects. Create a UG list consisting of the subgoals of the selected action, call TRLPlanner and update Status.

Insert ground effects :

For each effect do :

Find if there are entries of WDin which are possibly contradicted with the given effect, consulting the contradicted lists.

24 Artificial Intelligence in Engineering

```
findplan([], Status_in, Status_out).
findplan([G|UG], Status_in, Status_out) :-
    satisfyGoal(G, Status_in, Status_in1, [], NGrEout),
    findplan(UG, Status_in1, Status_out).

satisfyGoal(G, <Pin,WDin,Cin>, <Pout,WDout,Cout>, NGrEin, NGrEout) :-
    proveinTRLi(G, WDin, Cin, WD1, C1),
    findNonGroundEffects&GroundEffects(NGrEin, NGrEout, GrE),
    insertGroundEffects(GrE, WD1, C1, WDout, Cout),!.
satisfyGoal(G, <Pin,WDin,Cin>, <Pout,WDout,Cout>, NGrEin, NGrEout) :-
    selectAction(G, N, Cd, E, Cin, C1),
    append(NGrEin, E, Eff),
    findNonGroundEffects&GroundEffects(Eff, NGrEin1, GrE),
    expandAction(N, Cd, E,<Pin,WDin,C1>,<Pout,WDout,Cout>, NGrEin1, NGrEout, GrE).

select_action(<T1,T2>:Goal, <TA1,TA2>:ActN, Cd, [PE,GE], Cin, Cout):-
    action(<TA1,TA2>:ActN, D, Cd, [PE,GE], Cst),
    member(<T1,T2>:Goal, PE),
    check&appendConstraintsusingSCS([TA2 = TA1+D | Cst], Cin, Cout).

expandAction(N, [A,S], E,<Pin,WDin,Cin>,<Pout,WDout,Cout>,NGrEin1, NGrEout, GrE) :-
    proveinTRLi(A, WDin, Cin, WD1, C1),
    insertGroundEffects(GrE, WD1, C1, WD2, C2),
    findplan(S, <[N|Pin],WD2,C2>,<Pout,WDout,Cout> ).

insertGroundEffects([], WDin, Cin, WDin, Cin).
insertGroundEffects([E|GrE], WDin, Cin, WDout, Cout):-
    findContradicted(E, WDin, ContrEffs),
    maintainConsistency(E, ContrEffs, WDin, Cin, WD1, C1),
    append([E], WD1, WD2),
    insertGroundEffects(GrE, WD2, C1, WDout, Cout).

maintainConsistency(<T1,T2>:E, [<T3,T4>:CtrE | CtrEs], WDin, Cin, WDout, Cout):-
    check&appendConstraintsusingSCS([T2<T3], WDin, Cin, WD1, C1),!,
    maintainConsistency(<T1,T2>:E, CtrEs, WD1, C1, WDout, Cout).
maintainConsistency(<T1,T2>:E, [<T3,T4>:CtrE | CtrEs], WDin, Cin, WDout, Cout):-
    check&appendConstraintsusingSCS([T4<T1], WDin, Cin, WD1, C1),
    maintainConsistency(<T1,T2>:E, CtrEs, WD1, C1, WDout, Cout).
```

Figure 3. A prolog-like description of the planning algorithm

Use the SCS solver to maintain Status consistency by updating the Cin list with appropriate ordering constraints into the Cout list.

Update WDin with this effect into WDout.

In order to illustrate the execution of the planning algorithm, we have selected a simple blocks world example. We assume that the agent of the actions is a robot that moves blocks. We have not included any constraints that restrict the number of actions the robot performs at a time. Therefore the generated plan contains actions that can be parallely executed. If for example the robot is able

UG = [<TG1,TG2>:on(b,a), <TG3,TG4>:on(c,b)]

contradicted_lists([clear(X)], [holding(X),on(Y,X)], []).

contradicted_lists([holding(X)], [clear(X),on(X,Y),on(Y,X),ontable(X)], []).

contradicted_lists([ontable(X)], [on(X,Y),holding(X)], []).

contradicted_lists([on(X,Y)], [holding(X),holding(Y),clear(Y),ontable(X),on(X,Z)], [X≠Z]).

initial Status in :

WDin = [<T1,T2>:on(b,c), <T3,T4>:clear(b), <T5,T6>:clear(a), <T7,T8>:ontable(c)]

Pin = []

Cin = [T1≥s, T3≥s, T5≥s, T7≥s, TG2≤e, TG4≤e], /* s : start time, e: end time */

final Status out :

WDout= [<T1,T2>:on(b,c), <T3,T4>:clear(b), <T5,T6>:clear(a), <T7,T8>:ontable(c),
<T11,T12>:clear(c), <T13,T14>:holding(b), <T15,T16>:clear(b), <T17,T18>:holding(c),
<T19,T20>:clear(c), <TG1,TG2>:on(b,a), <TG3,TG4>:on(c,b)]

Pout = [<TA1,TA2>:unstack(b,c), <TA3,TA4>:stack(b,a), <TA5,TA6>:stack(c,b),
<TA7,TA8>:pickup(c)]

Cout = [T1≥s, T3≥s, T5≥s, T7≥s, TG2≤e, TG4≤e, T1<TA1, T2≥TA1, T11=TA2, T3<TA1,
T4=TA1+3, T13>TA1, T13≤TA2, T13<TA3, T14=TA3+12, T5<TA3, T6≥TA3, TG1>TA3,
TG1≤TA4, T15>TA3, T15≤TA4, T15<TA5, T16≥TA5, T7<TA7, T8≥TA7, T17=TA7+9,
T17<TA5, T18=TA5+12, TG3>TA5, TG3≤TA6, T19>TA5, T19≤TA6, T2<T11, T4<T13,
T6<TG1, T14<TG1, T14<T15, T12<T17, T16<TG3, T8<T17, T18<T19]

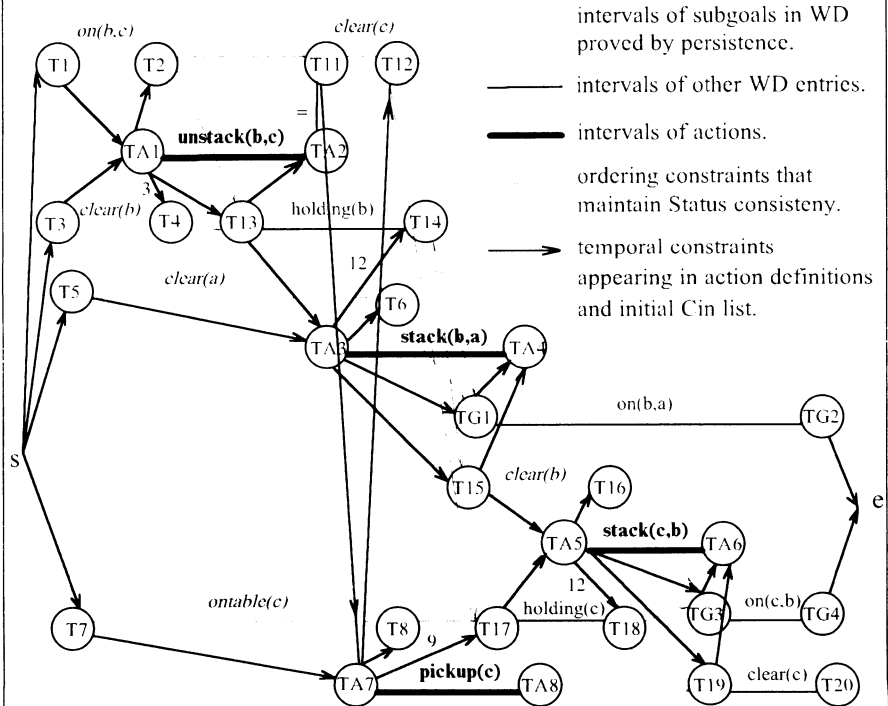


Figure 4. The generated plan of the blocks world example

26 Artificial Intelligence in Engineering

to perform only one action at a time (has just one hand), then this constraint will cause the generated plan to become linear. The definitions of the actions of the example are depicted in figure 2, while the initial Unsatisfied Goals list (UG), the contradicted_lists, the initial Status_in, the final Status_out and the time graph of the generated plan are depicted in figure 4.

This illustrated example could be assumed to be a subproblem either of a complicated assembly task where different objects of various schemes have to be put together or even of a robot motion problem where a mobile robot moves objects to remote locations.

5 Concluding Remarks

We have presented a general purpose planning algorithm which is strongly supported by the TRLi temporal logic system. The presented blocks world example is a very simple engineering problem. The ability of the planner to represent complex temporal constraints with such an expressive way, makes it useful to many real engineering applications. This happens because time is an important parameter in various phenomena and a factor for computing the ratio of change concerning machine functionality and behaviour. Time plays an important role to many practical or theoretical engineering issues.

The action representation schema that is proposed is more expressive than others. In Deviser [11] conditions necessarily persist until the end of the action, and effects start exactly at the end of the action. Although Triptic [10] is more expressive than Deviser, it is still more restrictive than the present approach, as it imposes default temporal constraints on actions. The relaxation of these default constraints is proposed as possible extension.

On the other hand, Allen [1] has proposed a system more expressive than others. Allen has liberated the temporal constraints between actions and their conditions and effects. Cheesman [2] integrates temporal reasoning with a planning procedure instead of using a separate temporal reasoning module, in order to have a system that operates efficiently. This has resulted in the development of a logic of relative interval relationships incorporated with the planning procedure. Both Cheesman's and Allen's systems are interval based and do not make use of quantitative temporal constraints.

The proposed formalism is expressive enough as it is liberated of any default temporal constraints and allows the user to impose any kind of reasonable temporal constraints dependent on the particular application. Moreover, the user can encode numerical quantitative information about the start and end of each temporal interval. Of course, when numerical information is included, the system becomes more efficient but its flexibility and generality is restricted.

Another innovation of the proposed planning algorithm, is the implementation of the corresponding action ordering as a temporal constraint solving problem. The ordering of actions is implied by the ordering of their temporal references. In this way, a plan network is not created and therefore the problems of its management are avoided.

We intend to extend the planning system to construct plans in a world where *external predicted events* occur. Such an event is specified as an action with a determined temporal reference. Thus the planner will be able to treat these events as it treats all other actions and reason about their interactions.

Acknowledgement

This work was fully funded by N.C.S.R. Demokritos. At that time Prof. T. Panayiotopoulos was working as a research fellow at the Institute of Informatics and Telecommunications of N.C.S.R. Demokritos.

References

1. Allen, J.F. Planning as Temporal Reasoning, in KR'91 (ed J.Allen, R.Fikes, E.Sandewall), pp. 3-14. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, 1991.
2. Cheesman, P. A Representation of Time for Automatic Planning. *IEEE International Conference on Robotics*, 1984, 513-518.
3. Currie, K. & Tate, A. O-PLAN: the open system architecture. *Artificial Intelligence* 52, 1991, 49-86.
4. Dean, T. & McDermott, D. Temporal Data Base Management. *Artificial Intelligence* 32, 1987, 1-55.
5. Drabble, B. & Kirby, R. Associating A.I. Planner Entities with an underlying Time Point Network, in EWSP '91, (ed J. Hertzberg), pp. 27-38. *Proceedings of the European Workshop on Planning*, Sankt Augustin, FRG, 1991.
6. Fikes, R. & Nilsson, N. STRIPS: A new approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2, 1971, 189-208.
7. Panayiotopoulos, T. & Spyropoulos, C.D. TRL: A formal Language for Temporal References, ICTL Workshop, (ed H.J.Ohlbach), pp.99-109. *Temporal Logic Proceedings of the ICTL Workshop*, MPI-I-94-230, Bohn, Germany, 1994.
8. Panayiotopoulos, T. & Gergatsoulis, M. A Prolog like temporal reasoning system, *12th IASTED International Conference on Applied Informatics*, Innsbruck, Austria, Feb.21-23, 1995.
9. Panayiotopoulos, T. & Gergatsoulis, M. Intelligent Information Processing using TRLi, submitted for presentation.
10. Rutten, E. & Hertzberg, J. Temporal Planner = Nonlinear Planner + Time Map Manager, *AI Communications*, Vol.6, Nr.1, 1993, 18-26.
11. Vere, S.A. Planning in Time: Windows and Durations for Activities and Goals, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-5, No.3,1983, 246-267.