The Indexed Time Table Approach for Planning and Acting

Malik Ghallab and Amine Mounir Alaoui
LAAS-CNRS
7, Av Colonel Roche, 31077 Toulouse, France

Abstract:

We are interested here in a representation of symbolic temporal relations, called **IxTeT**, that is both powerful enough at the reasoning level for tasks such as plan generation, refinement and modification, and efficient enough for dealing with real time constraints in action monitoring and reactive planning.

Section 1 argues that such representation for dealing with time is needed in a teleoperated space robot. After a brief survey of known approaches, section 3 introduces the proposed representation, and shows its computational efficiency for managing a large data base of temporal relations. Reactive planning with IxTeT is described in section 4 and exemplified through the problem of mission planning and modification for a simple surveying satellite.

1. Introduction

This paper addresses the problem of real-time acting and reacting for robot in a dynamic environment, at the level of representation, reasoning, and decision making aspects.

Such problems arise in robotics applications that are either too complex and changing to allow complete and reliable hand programming, or that require some level of autonomy. Most space robotics applications are in both cases. Applications in *unstructured environments*, such as robots for planetary exploration, will require a high level of autonomy in order to cope with a large set of tasks in a broad spectrum of conditions. Robots in *structured environments*, such as space stations, should also be versatile, even if less autonomous. In both cases, teleoperation is complementary to and does not contradict decision making abilities, intelligence, and autonomy in tasks ranging from sensing and environment interpretation to planning and acting. This is more true for real-time tasks such as reacting to unexpected events. Indeed, manned control of a space robot should be thought of mainly as providing the system with goals, eventually decomposed into a flexible structure of subgoals and tasks, that are programming the robot at the task level.

Plans, either given or autonomously generated, need refinement or revision at execution time. Conditional plans involve choices between alternatives, eventually only one of which (the most likely one) has been fully pursued at planning time but another one may need to be considered at execution time. Unexpected events may require partial modification or rejection of the current plan. Goals may have to be discarded or postponed for more urgent ones.

Thus in addition to planning, acting and reacting means: monitoring and keeping track of the state of achievement of current plan, refining adequately actions and implementing them, choosing from alternatives, deciding about short term reactions to unexpected events. Those tasks involve mainly 2 temporal aspects: how one represents time and reasons on temporal relations (e.g. for keeping track of the present time), and how one deals with real-time.

Any action involves time as a particular resource. A good representation should exhibit the rich temporal structure relating an action to its effects, and to expected events and pursued goals. In that sense, monitoring is keeping track of the present time relatively to the projected future. The advance of time is discontinuous and driven by asynchronous events (although a robot may have a clock that gives it the absolute time). The happening of an expected event instanciate one particular future among the alternatives planned.

The real-time requirement is mainly due to the dynamic nature of the environment, *i.e.* to unexpected changes and events that happen asynchronously, are not under the robot control, but that require from it adequate reactions. Real-time reaction to asynchronous events means an a priorily bounded response time, at least for the first of a hierarchy of actions, such as:

- immediate reflex action,
- short term adaptation to the situation (for assessment, goals evaluation and replanning),
- long term reaction according to the new plan.

For the problem considered we thus need a representation of time and temporal relations that is both powerful from the reasoning point of view and efficient from the computational point of view.

After a brief survey of known approaches, section 3 introduces the proposed representation, called IxTeT, and shows its computational efficiency for managing a large data base of temporal relations. Reactive planning with IxTeT is described in section 4 and exemplified through the problem of managing a simple surveying satellite.

2. State of the Art

Classical situation calculus and state-space representations consider actions as instantaneous transitions. They cannot represent time or forthcoming events not resulting from the planner's activity.

Procedural or tasks networks [11][13] suffer from the same limitations, although they deal with partially ordered flexible plans. Some extensions, such as that of DEVISER [15], add a numerical representation of time, like durations and bounds (windows) on activities, that are managed by a mixing of Operations Research (e.g. PERT-chart) and AI techniques. However symbolic temporal representations are not allowed by such approaches, they cannot deal with relative relationships (e.g. between goals, events, actions and their effects), and hence offer very restricted reasoning capabilities.

Three approaches can be used for a symbolic representation of time:
- time as a term of a predicate in classical logic: a limited representation;
- time as a modality in temporal logic: seems to be very complex for planning tasks;
- time as an element in couples <logical formula F, temporal qualification of F> : this "reified logic" approach [9], [3], and its extensions [11], was found to have some nice properties. It manages separately the temporal qualifications through a so-called *Time-Map Manager* (TMM) that is in charge of retrieval and updating of a knowledge base of temporal relations. In a typical application, a TMM will be put at a very low level and in heavy use, it has to be very efficient.

Of this last approach, the *Algebra of Temporal Interval* [1] is the most popular representation: it is appealing for its ease of implementation and expressive power in planning tasks [2]. It has however a major drawback: the consistency problem for a set of Interval Algebra relations was proven to be NP-Hard [16]. But of resorting to exponential algorithms, this leads to the use of a transitive closure propagation algorithm that is defective because of:
- a completeness problem: it may accept an inconsistent set of relations as being consistent; and
- a complexity problem: it runs in $O(n^3)$, a too high complexity for large applications.

As it was argued in [16] one can solve the completeness problem of this algorithm by restricting the expressive power to a sub-class of Interval Algebra. That class is equivalent to the *Time Point Algebra*. For those two representations we are proposing a complete and much more efficient method than the transitive closure propagation algorithm.

3. Managing a Time Map with IxTeT

A natural representation for a set of temporal relations is a network where nodes are time tokens (i.e. intervals or instants) and arcs are labeled by the constraints relating two nodes. Two directions can be pursued for performing the two tasks of:

(i) retrieval (whether and how two events are related) and

(ii) updating (add new events and temporal relations)

- either using a complete graph where all possible relations between all pairs of nodes are propagated and explicitly maintained: this makes retrieval trivial in O(1), and requires a costly propagation algorithm in $O(n^3)$ for the updating task;
- or using a network where the only arcs are those of the explicit knowledge of the problem: this simplifies updating but requires for retrieval a costly search through possible paths of the network.

The approach proposed here is a trade-off between these two directions. It relies on the efficient combination of 2 principles:
- adding to the time-network a particular data structure, a maximal spanning tree with an adequate indexing scheme, that permits the computation of ancestral information in O(1), this greatly simplifies task (i), and

- restricting the propagation of new relations to a small subset of nodes in the network in order to perform task (ii) efficiently.

In a time-point algebra 3 elementary relations, *before*, *equal* and *after*, and their 5 disjunctive combinations relate a finite set of instants or time-points. Instead of a network with arcs labeled by relations, we use 2 different types of unlabeled arcs:
- arc $\langle$ standing for the relation (*before* or *equal*), and
- arc $\neq$ meaning the relation (before or after).

The 8 possible relations between 2 time-points are easily expressed as 0, 1 or 2 arcs relating two nodes.A network of $\langle$ and $\neq$ arcs corresponds to a consistent set of relations if no pair of nodes, connected by a $\neq$ arc, are involved in a loop through $\langle$ arcs. Such a loop describes a set of identical time-points that should be collapsed to a single node. Individual events corresponding to this set are kept distinct but their simultaneity is recorded by connecting all of them to the same node in the time-map. Arcs $\neq$ do not require any propagation mechanism; they are looked for only when a collapsing decision has to be taken. For that reason we can keep arcs $\neq$ implicit in the network representation.

A consistent network where all possible collapsing operations have been performed contains only $\langle$ arcs and is loop-free. It thus defines a partial order over the set of nodes. Since we can always add for convenience an origin time-point, we endup finally with a network that is a rooted DAG, *i.e.* a time-lattice.

Let us denote it L=(U,A) where: U= $\{t_0, t, u, v, w, ...\}$ is the set of time-points, t_0 being the root of L; and A is the set of $\langle$ arcs in L.
Point u precedes temporally (is *before* or *equal*) point v if there is a path in L going from u to v. Let us denote u « v this fact (« is the transitive closure of $\langle$). Thus relating 2 points requires a search of a path in L . How can we speed-up such a search?

3.1. Representation

To speed-up this search we use the fact that ancestral information can be computed in constant time for a tree correctly ordered. Two problems should be addressed: (1) classical tree ordering is not easily updated and maintained for a dynamically growing structure, and (2) how to map a time-lattice to a tree.

We solve this last problem by extracting from L a maximum spanning tree T defined as follow:
- T is rooted at t_0 (it is not a free tree as is usually the case for a spanning tree), and covers all nodes of L;
- the number of arcs in T is maximal.

Let us denote by r(u) the rank of u in L, *i.e.* the length of the longest path in L from t_0 to*u*:

$r(u)= 1 + \max \{r(v)/ \forall(v,u) \in A\}$, with $r(t_0)=0$. To compute T from L we first order the nodes in L according to their rank. This can be achieved by an O(|A|) breadth first search in L: all successors of nodes of rank k have their rank set to k+1, which may change previously computed ranks if longer paths are found (some simple additional tests speed-up the procedure).

Let M be the maximal rank found in L. We start from any node z of rank M, put it in T, choose among its predecessors in L any node y of rank M-1 and put in T the arc (y,z) and the node y as the parent node of z: p(z)=y. This is repeated for a predecessor x of y such as r(x)=r(y)-1; arc (x,y) and node x are added to T. We keep on moving up along a path of maximal length until the root t_0 is put in T. The procedure is repeated starting from a node of maximal rank among those not already in T. While processing node u if there is a choice between several of its predecessors, all at rank r(u)-1, we choose one not already in T and add it to T. If none remains we choose among such predecessors one in T which has the least number of children in T, and attach a new path to the spanning tree. The procedure is repeated until all nodes of L are put in T.

Let s(u) be the set of children of u in T, and s*(u) the set of its descendants in T (transitive closure of s). To test whether $v \in s^*(u)$ efficiently we attach to each node u as index a sequence I(u)=(i1 i2 ... ik) of one or more integers that is defined, while generating T, as follows:
- nodes of the path (t_0, ..., x, y, z) that was first put in T are indexed by their rank:
I(z)=(M), I(y)=(M-1), I(x)=(M-2), ... , I(t_0)=(0);

- if I(u)=(i1 i2 ... ik) and v $\in$ s(u) then:
  - if |s(u)|=1 (v is the first children of u in T) then I(v)=(i1 i2 ... i(k-1) (ik+1))
  - if |s(u)|=2 then I(v)=(i1 i2 ... ik 1)
  - if |s(u)|>2 then I(v)=(i1 i2 ... ik (3 - |s(u)|) 1)

Nodes of the first path put in T can be indexed while they are added to T. The other nodes are not indexed until their path is attached to T: indexing proceeds by moving from the attachment parent (already indexed) down along the path. Notice that the indexing is easily maintained for a dynamically growing structure: the addition of a new node (as a leaf) in the tree does not change the indexing of the previous node.

The main property of this indexing scheme is the following:
if I(u)=(i1 i2 ... ik), and I(v)=(j1 j2 ... jh) then
v $\in$ s*(u) iff $k \leq h$, (i1 i2 ... ik-1)=(j1 j2 ... jk-1) and ik $\leq$ jk (1)

Thus, to relate 2 nodes u in v in T we first compare their rank
- if r(u)=r(v) then u and v are not related in T;
- if r(u) < r(v) : either condition (1) is satisfied: v is a descendant of u, or they are not related;
- if r(u) > r(v) : u and v are permuted before checking condition (1).

Notice that the rank of a node is given by its index : $r(u)=\Sigma u_i$; for i=1 to k, and $u_i > 0$

A precedence relation in L, such as u « v , can be retrieved either
- through the spanning tree T if v $\in$ s*(u) : this is checked easily; or
- through a path using some arcs of L not belonging to T.

To take care of this last case, arcs not in T are processed by 2 operations:
(1) Eliminating redundant arcs: if (u,v) $\in$ A is an arc of L not belonging to T such as v $\in$ s*(u) then this arc does not bring any useful information. It is redundant and can be eliminated.
(2)Propagating non redundant arcs: 2 nodes may be linked by a path that mixes in any order arcs from T and non redundant residue arcs. To avoid mixing the 2 structures and simplify the retrieval of precedence relations we propagate recursively a non redundant arc (u,v) to the parent node of u in T, unless v is already a descendant of p(u) in T, *i.e.* if v $\in$ s*(p(u)).

The procedure corresponds to the trade-off mentioned earlier between using a complete graph and keeping a minimal set of relations. Let us call *residue arcs* the set obtained after elimination and propagation.The important property here is that any residue arc (u,v) is such that r(u) < r(v). This is trivially true for arcs in L not belonging to T, it is also true for added arcs since the propagation goes only upward in T.

3.2 Algorithms

To make clear the distinction between the part of the time-lattice L covered by the spanning tree T and the residue part:
- in T we will speak of the *parent* p(u) of a node u, its *children* s(u), its *descendants* s*(u) and its *ancestors*; I(u) is its index and r(u) its rank; only s, p and I are kept as data structures.
- in the residue part, a(u) denotes the set of nodes linked to u by residue arcs $\langle$ .We will speak of the *followers* and *foregoers* of a node for adjacency relations defined by such arcs (reserving *successors* and *predecessors* for the complete lattice).

Notice that s(u) $\cap$ a(u)=$\varnothing$: the successors of a node are partitioned into its children and its followers.

3.2.1. Retrieval

A precedence relation in L is characterized by:

$$u \ll v \Leftrightarrow \quad (r(u) < r(v)) \\ \wedge [(v \in s^*(u)) \vee (v \in a(u)) \vee (\exists\, w \in a(u) / w \ll v)]$$

The 3 first conditions come from the fact that u precedes v in L if v is a descendant of u in T or if it is a follower of u. The last one is due to the property of the upward propagation mechanism: all followers of the children of u are either the followers of u or its descendants; there cannot be a non descendant node of u linked to u through s*(u) that is not also linked to u through a(u).

The comparison algorithm between two points u and v is thus:

```
Compare (u, v)
    If r(u)=r(v) then return(nil)
    else   if r(u) > r(v) then Relate (v, u)
           else Relate (u, v)

Relate (u, v)
    if v ∈ s*(u) or v ∈ a(u) then return (u « v)
    else   for each w ∈ a(u)
               if [r(w) < r(v) and Relate(w, v) return (w « v)
                   then return (u « v)
           return nil
```

Notice that the recursive calls to Relate are pruned when the rank of w reaches that of v

3.2.2. Updating

Adding new points and relations should be done such as to keep all the properties of the representation. The addition a point *w* and 2 relations with *u* and *v* (u ⟨ w ⟨ v) can be decomposed into 2 steps:
- add *w* as a child of *u* and give it the right index ; and
- add an arc between *w* and *v* and update the tree and residue arc if necessary.

The first operation is straightforward. The second operation involves 3 steps: a test, and eventually a propagation and a reindexation.

The test determines whether v « w, in this case the updating is impossible (w ⟨ v can be inserted) unless all points in every path from *v* to *u* can be collapsed.

The reindexation takes place if r(w) ≥ r(v). In this case, to keep *v* on the longest path in the spanning tree, we give to *v* a new parent node *w*. Node *v* is removed from the children of p(v) and put as a follower of p(v); p(v) becomes *w*; the descendants of *v* are reindexed. The reindexation procedure computes the new index of each node according to the index of *v*, it then verifies if the followers of *v* have the right rank considering the new index of *v*, and, if not, it reindexes them. This is repeated recursively.

If r(w) < r(v), v is put as a follower of *w*. This residue arc is propagated to the parent of *w*, and recursively to its ancestors that are not found by procedure **Relate** linked to *v*. Notice that if there is a reindexation, there will be propagation of the arc between the old parent of *v* and *v*: all the former ancestors of *v* have to know that they are still linked to it.

3.3. Performances

Reindexation and backward propagation procedures are detailed in [5], together with the 2 other tasks performed by the TMM : collapsing and removal. This reference also reports on experimental results for a set of time lattices of size up to 2000 points. On 60000 runs the IxTeT Time-Map Manager exhibits a linear complexity for both operations, retrieval and updating. The linearity constant is fairly low: in a 2000 points lattice about .5 second is required to insert a new point and 2 relations and less than .05 second to compare 2 points (for a non optimized Lisp implementation on a Sun3). The space complexity of IxTeT seems also to grow linearly.

IxTeT abilities for dealing with symbolic temporal knowledge can be extended to numerical quantitative constraints and to constant points (dates). All numeric relations and dates may be given with intervals precising earliest and latest possible occurrence. It is easy to combine in IxTeT an absolute referencing system by dates with qualitative relations. A point can be attached to a date. Arcs can be labeled by durations. The direct link that exists between durations and dates allows several deductions on the order of points in the lattice. A lattice may have some points known as precise dates, others are variables linked with qualitative relations, and others have dates deduced from quantitative relations.

4. Planning with IxTeT

The *IxTeT* representation for planning relies on a 2-dimensional array with rows corresponding to logical assertions and columns to time points (instants). Cells in the table are temporal qualifications of the assertions. Instants are related through a time lattice that is managed as described earlier.

The user defines in a declarative way a set of decomposition operators, each one being a description of the steps needed to achieve an elementary task. This description gives recursively the actions, subtasks, conditions required, and goals achieved by the task, together with their temporal links. Those are stated as symbolic relations over *implicit intervals* using elementary relations of interval algebra (start, meet, finish, overlap, before, during equal and their inverse). Indeed, it is preferable and easier to express input temporal knowledge in terms of intervals, and to transform it for efficiency reasons into time points. Actions are defined separately from the operators through their effects, temporally linked to the action. One may specify an action with a duration and effects that take place when the action starts, when it is going on, when it finishes, or effects that are later delayed.

By actions, we mean here the elementary actions that cannot be divided into smaller parts. Of course, each action may have several arguments that define its possible parameters. So the description of an action will be:

```
(Describe (Action arg1 arg2 ... argn)
        (Effects (effect1)
                 (effect2)
                 ...
                 (effectk))
        (Duration (d)))
```

Each effect in the description is, in fact, the name of one of the effects induced by the action preceded by a temporal relation defining precisely when the effect takes place *vs* the action. As we said, the temporal relation is one of the 13 relations of interval algebra. The duration may be defined by a constant number or by a function of the arguments. A **When** field may be added to the action to define the context when the action can be performed.

When elementary actions are defined, it is possible to combine them into operators that achieve a task. A task will be a combination of several actions linked by temporal relations and some time constraints. Time constraints are here to define the environment in which the task can be realized. All this is defined as follows:

```
(To Achieve (Task arg1 arg2 ... argn)
        (TimeCond (TC1)
                 (TC2)
                 ...
                 (TCk))
        (Do Steps
                 s1 (Action1)
                 s2 (Action2)
                   ...
                 sp (Actionp))
        (Such That (TempRel1)
                 (TempRel2)
                 ...
                 (TempRelq)))
```

The **TimeCond** defines all the temporal constraints of the task: it cannot begin before certain effects are verified, or it has to keep something true during the whole execution, etc... The constraints are defined by one or more of the 13 interval relations followed by an effect. If there is more than one relation, the set of relations is disjunctive and taken in the subset of restricted interval algebra that is compatible with the time points algebra.

The **Such That** field defines the temporal relations that link the actions performed in the task. With this, the task is totally defined with its constraints and the actions it takes (and so, the effects it induces).
The implicit intervals in this user-defined knowledge and their relations are automatically translated into a lattice of time points. This is done by *compiling* each task operator into a general *IxTeT* (*i.e.* assertions have quantified variables that are consistently related). A plan with such representation is an instantiated *IxTeT*.

The *compilation* is an off-line preparation for the planner: this operation creates the time lattice corresponding to each task, attaching to the beginning or the end of effects or actions a time point. The lattice is easily created by the temporal constraints contained in the task, and its coherence is tested (for instance, an action cannot have two beginnings and no end). So the result is in two parts: a table giving for each action and effect its beginning(s) and end(s), and a time lattice relating those points.

This preprocessing work simplifies the work of the planner: it has not to build all the lattices and to test their coherence, it has only to work with ready parts to build the plan. The *compiled* task can be inserted in any plan (represented itself by a lattice and a table) just like a brick in a wall. This is very important because it allows fast insertion of tasks in a reactive system: if the system is fast enough, it is possible to change the plan during its execution according to the changes of external world.

Planning, along the hierarchical goal decomposition approach proceeds by attaching tasks to goals and subgoals. This corresponds here to the insertion (and instanciation) of new ***IxTeT***s into the current plan. Controlling the plan execution while acting is performed by progressively reducing the time-lattice to an ordered sequence as time advances. This results either from an opportunistic choice made at execution time, or through the observed occurrence of an expected event. Reacting to unexpected events on the other hand involves considering new subgoals, *i.e.* inserting new ***IxTeT***s into the projected future, and reconsidering the need of previously planned actions an goals.

Let us illustrate some of the planning processes with IxTeT through the example of managing a simplified earth observation satellite similar to SPOT[8]. There are two cameras on board that may be used either each one alone or in conjunction. Each camera may have its mode (panchromatic or multispectral) and its focal distance changed and a mirror in front of it moved such as to aim at a particular direction.

On board, there are two recorders that allow images to be kept if necessary until they are sent to earth. Each recorder may be freely used by any of the two cameras. And there is also a radio system allowing reception of requests from earth and sending of images.

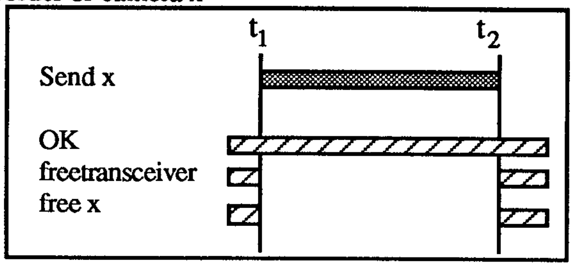
Images of a region on earth can be taken only under certain conditions. These conditions can be summarized by parts of the trajectory in which photographs of the region can be taken. So the whole trajectory of the satellite will be cut into numbered parts.

In the IxTeT formalism, we can attempt now to represent some of the most important actions and tasks of satellite:

Elementary Actions

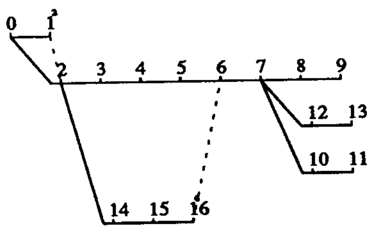
```
(Describe (ChangeFoc x a b)                 ; Change focal dist. of camera x
        (Effects                            ; from an initial value a to b
                (mi foc x a)
                (m foc x b))
        (Duration (+ 10 (* 5 (abs (- b a))))))      ; Notice that duration is a function here

(Describe (LockFoc x)                       ; Lock focal distance of camera x
        (Effects (m lockedfoc x))           ; to its current value
        (Duration 1))                       ; Here, duration is a constant

(Describe (UnlockFoc x)                     ; Unlock focal of camera x
        (Effects
                (mi lockedfoc x))
        (Duration 1))

(Describe (ChangePos x a b)                 ; Change position of mirror x
        (Effects                            ; from an initial position a to b
                (mi pos x a)
                (m pos x b))
        (Duration (+ 10 (* 5 (abs (- b a))))))

(Describe (LockPos x)                       ; Lock position of mirror of camera x
        (Effects (m lockedpos x))           ; to its current value
        (Duration 1))
```

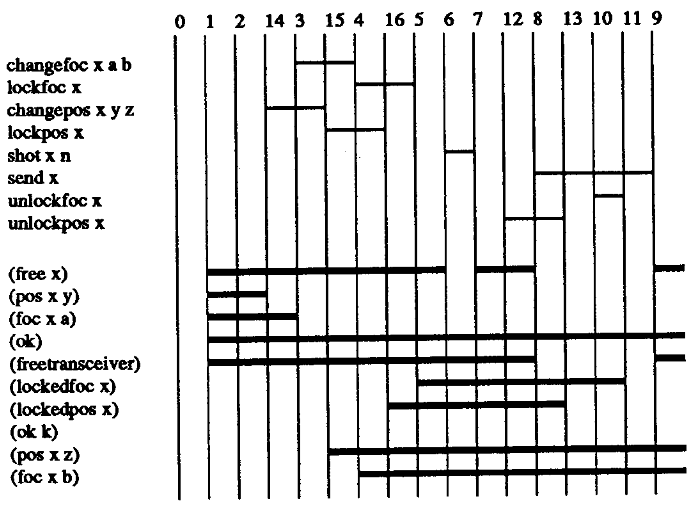
```
(Describe (UnlockPos x)                 ; Unlock position of mirror of camera x
        (Effects
                (mi lockedpos x))
        (Duration 1))

(Describe (Record x y)                  ; Record image in camera x in
        (Effects                        ; recorder y
                (mi free x)
                (mi free y)
                (m free y)
                (m free x))
        (Duration 10))

(Describe (Shot x n)                    ; Get images from camera x during
        (Effects                        ; a period n (the mode may be added)
                (mi free x)
                (m free x))
        (Duration (+ 15 (* n 5))))

(Describe (Send x)          ; Send image from recorder or camera x
    (Effects
        (d OK)
        (mi freetransceiver)
        (mi free x)
        (m free x)
        (m freetransceiver))
    (Duration 100))
```



Tasks performed by the satellite

```
(To achieve (ShotAndSend k x n a b y z)   ; Get images with camera x during n
        (TimeCond                         ; and from the trajectory part k
                (When free x)             ; with focus b and mirror position z
                (After lockedfoc x)       ; Then send the image
                (After lockedpos x)
                (While OK k))
        (Do Steps
                s1 (changefoc x a b)
                s2 (lockfoc x)
                s3 (changepos x y z)
                s4 (lockpos x)
                s5 (shot x n)
                s6 (send x)
                s7 (unlockfoc x)
                s8 (unlockpos x))
        (Such That
                (m s1 s2)
                (m s3 s4)
                (< s4 s5)
                (< s2 s5)
                (< s5 s6)
                (< s5 s7)
                (< s5 s8)))
```

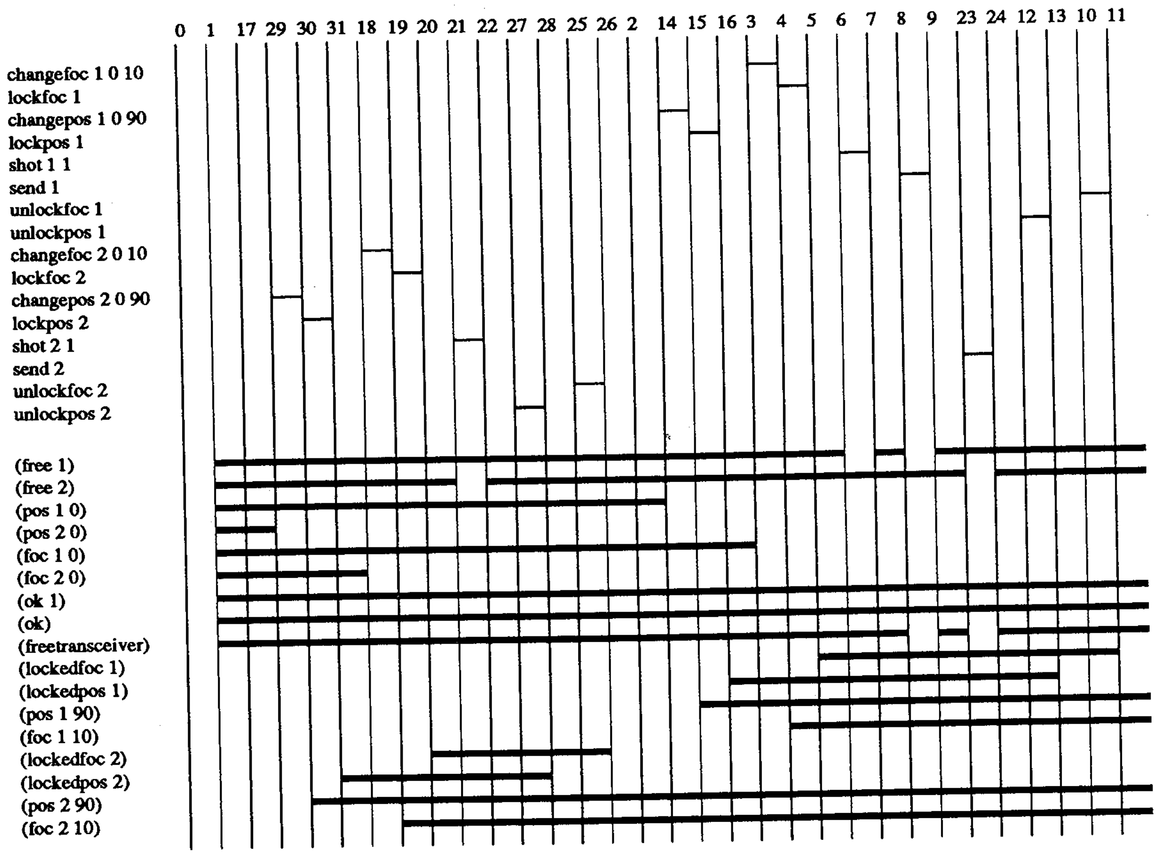


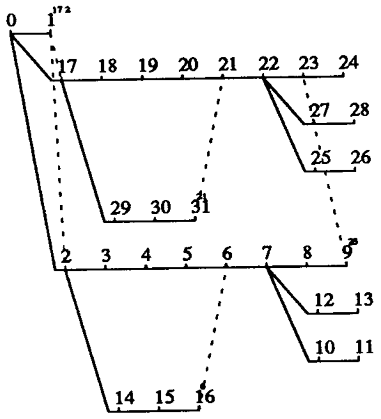


Notice that the **TimeCond** field contains time constraints other than those of interval algebra. This is because it is much more simple to have the single word **When** than (mi oi f d). The disjunction (mi oi f d) is a constraint which is compatible with the time points algebra [6].

Notice that all the constraints are relative, but durations and some absolute dates may give to the time lattice other constraints that are not incompatible with the first ones. In fact, a mixed system with absolute and relative constraints is completely representable with this formalism.

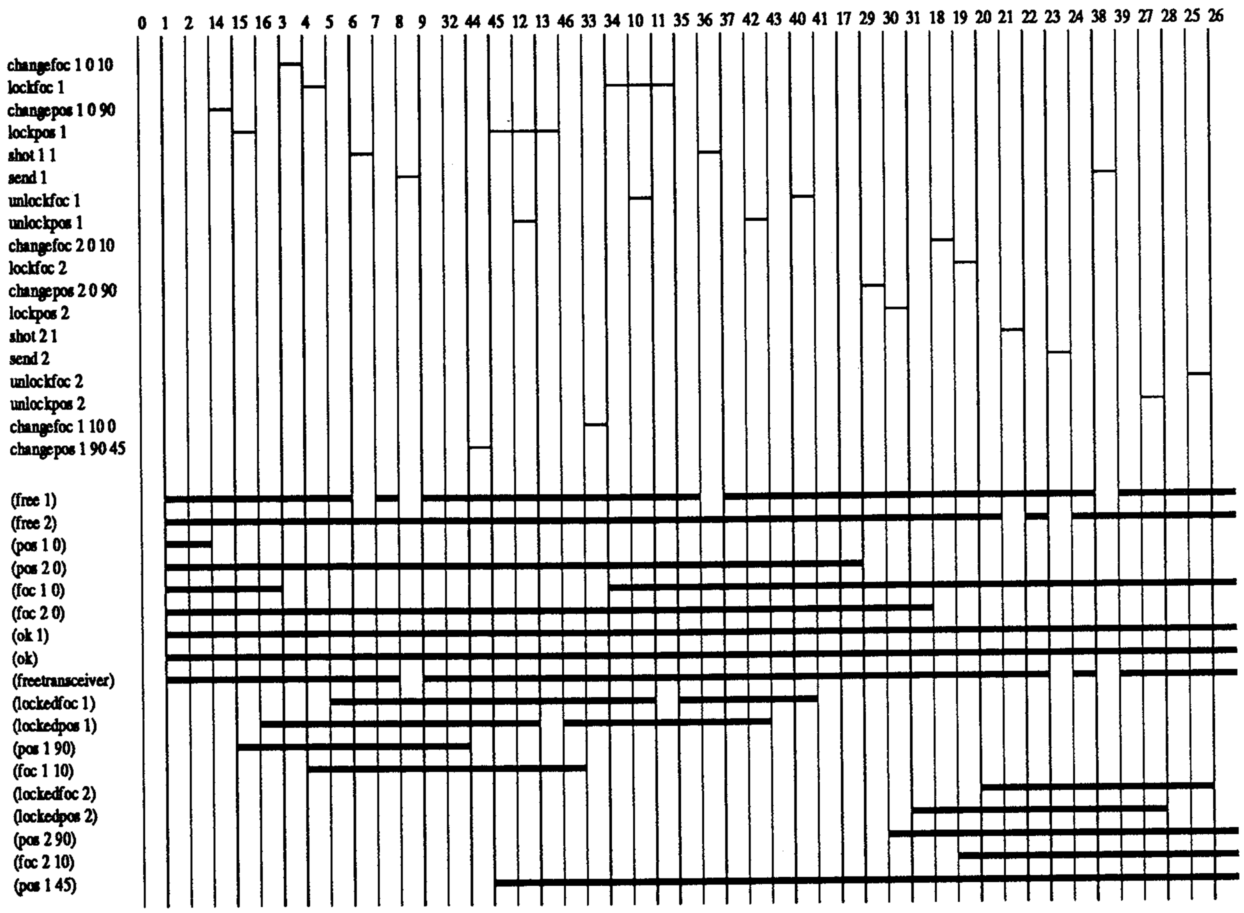
Now, it is possible from this example to build some planned actions. For instance 2 tasks **ShotAndSend** may occur one after the other with different arguments, and then we want to insert a new one. We see in the generated lattice (figure below) all the constraints that the planner has to deal with; but we see also all the time points that are independent and that represent the possible parallelism between actions.

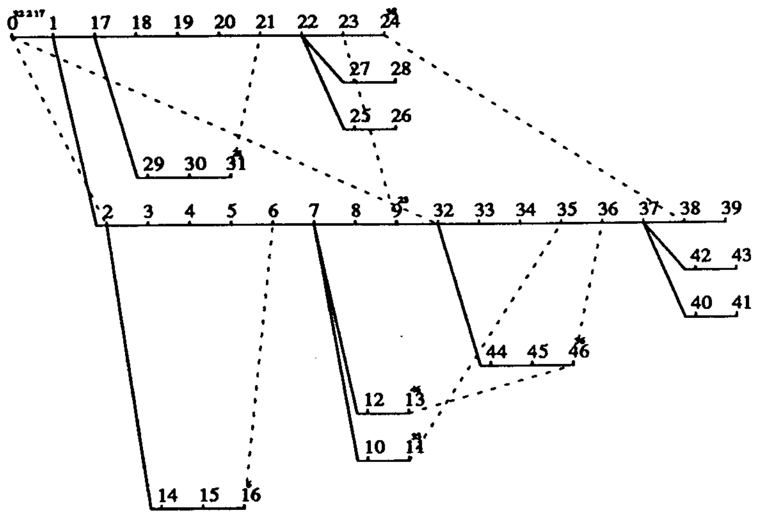




The previous figure shows the combined actions (Shotandsend 1 1 1 0 10 0 90) and (Shotandsend 1 2 1 0 10 0 90). The two tasks are almost completely independent: the only constraint between them is the use of the transceiver to transmit data to earth. This is represented by the constraint t9 ⟨ t23.

Insertion





Let us now add a new task with camera 1: (Shotandsend 1 1 1 10 0 90 45). This new task will be placed opportunistically with respect to the constraints induced by each action and the already computed IxTeT. We see clearly that this task is very constrained by the former task with the same camera, but only one constraint attaches it to the task with the other camera: the use of the transceiver.

This example shows clearly the IxTeT ability for dealing with temporal constraints, placing them opportunistically at the right place without any other information than the temporal constraints given to it. Another point is very important: this is done quickly. The compilation of the set of tasks took 1.8 second and the insertion of the third task in the final plan about .16 second (on a Sun 3). The response time of IxTeT is low enough to permit its use in real-time reactive systems.

This simple example did not illustrate all aspects of the IxTeT planning system. The actions and tasks may affect more than the simple state of the world: they may change the reasoning process by some side effects that are not directly induced by the tasks. Sometime, it is necessary to guide the planner in the choice of the task to be performed. All this is done by rules. There are two types of rules: temporal and non-temporal rules.

The non-temporal rules describe the logical relations that may exist between effects. For example, if a camera is getting an image, it is not sending images to the recorders. These rules describe essentially all the exclusions between actions and effects, and so, the side effects of actions and tasks. They enable to compute the qualification and ramifications of tasks.

Temporal rules are essentially rules to induce tasks in certain temporal situations. For example, if the satellite will be over a certain region for more than 1 minute, photos have to be taken. Such rules are slightly different from the ones we saw before: their conditions are temporal relations, and their second part contains tasks and not effects. The use of this second type of rule is essentially to give the planner a priority in its choice of a task.

5. Conclusion

This paper has argued that teleoperation for a space robot is complementary to the autonomy needed in tasks ranging from sensing and environment interpretation to planning, acting and reacting to events. Those tasks involve time as a constraint and as a concept to be explicitly represented and reasoned with. One thus needs a representation of time and temporal relations that is both powerful from the reasoning point of view and efficient from the computational point of view.

A representation aiming at such goals has been proposed. It relies on the time point algebra for expressing symbolic temporal relations. A data base of such relations is managed through an efficient data structure, an indexed maximal spanning tree of the time lattice. We have shown how this structure is built, maintained and used by the Time Map Manager developed. Empirical evidences have been reported that support the linear complexity of the algorithms involved and the overall efficiency of the proposed representation.

The use of the proposed representation in planning was thus considered. A formalism for describing elementary actions and tasks in terms of symbolic temporal relations was developed. A preprocessing transforms such description of tasks into particular arrays and time lattices. Planning proceeds along the hierarchical goal decomposition approach by inserting such arrays and time lattices into a larger similar structure of goals, subgoals and expected events. Plan refinement and modification are carried out by the same process.

This planning system has been exemplified through the task of generating and changing the mission plan of a simple surveying satellite.

References

[1] Allen, J. F. Maintaining knowledge about temporal intervals Communications of the ACM 26(11):832-843, November 1983
[2] Allen J.F and Koomen J.A. Planning using a Temporal World Model. Proc. 8th IJCAI, Aug. 1983
[3] Allen, J. F. Towards a general theory of action and time. Artificial intelligence 23: 123-154, 1984
[4] Ghallab, M., Alami, R. and Chatila, R. Dealing with time in planning and execution monitoring Robotics Research 4, R. Bolles, MIT Press, 1988

[5] Ghallab M. and Mounir Alaoui A. Managing efficiently temporal relations through indexed spanning trees. Rappot LAAS 88.360, Dec. 1988, 13p.

[6] Granier, T. Contribution a l'etude du temps objectif dans le raisonnement Rapport LIFIA RR 716-I-73, Grenoble, February 1988

[7] Koomen, J. A. G. M. The TIMELOGIC temporal reasoning system in common lisp Technical report TR 231, Rochester University, November 1987

[8] Maarek, G., Gateau, D., Hua, L. and Proth, J.M. Projet SPOT, Rapport final CNES/INRIA/SEMA-METRA, September 1987

[9] McDermott, D. V. A temporal logic for reasoning about processes and plans Cognitive Sci. 6:101-155, 1982

[10] Mounir Alaoui, A. IxTeT: un systeme de gestion de treillis d'instants Rapport LAAS 88.154, Laboratoire d'Automatique et d'Analyse des Systemes / CNRS, Toulouse, June 1988

[11] Sacerdoti E.D. A Structure for Plan and Behavior. Elsevier, 1977

[12] Shoham, Y. Temporal logics in AI: semantical and ontological considerations Artificial intelligence 33: 89-104, 1987

[13] Tate A. Generating Project Network. Proc. 5th IJCAI, Aug. 1977

[14] Tsang, E. Time structure for AI in Proceedings of the tenth IJCAI: 456-461, 1987

[15] Vere S.A. Planning in Time: Windows and Durations for Activities and Goals. IEEE Trans. PAMI, 5, 3, May 1983

[16] Vilain, M. and Kautz, H Constraint propagation algorithms for temporal reasoning in Proceedings of the fifth national conference on artificial intelligence (AAAI-86):377-382, August 1986

Acknowledgement: This work was supported by the EEC under the ESPRIT project P1560 (SKIDS), and by the French National Research Program GRECO/PRC-IA.