2010 17th International Symposium on Temporal Representation and Reasoning

Solving Temporally-Cyclic Planning Problems

Martin C. Cooper Frédéric Maris Pierre Régnier
IRIT, Université Paul Sabatier,
118 route de Narbonne
31062 Toulouse, cedex 9, France
{cooper, maris, regnier}@irit.fr

Abstract—**In order to correctly model certain real-world planning problems, it is essential to take into account time. This is the case for problems requiring the concurrent execution of actions (known as temporally-expressive problems). However, we show in this paper that certain existing planners which solve this type of problem are, in fact, incomplete. They cannot guarantee to find a solution to a problem involving sets of cyclically-dependent actions (which we call temporally-cyclic problems). We characterize those temporal planning languages which can express temporally-cyclic problems. We also present a polynomial-time algorithm which transforms a temporally-cyclic problem into an equivalent acyclic problem. Applying our transformation restores the completeness of these temporal planners.**

I. INTRODUCTION

An important challenge for today's planners is the management of the time dimension, to allow the synchronisation of non-instantaneous actions. A large majority of real-world problems, in order to be solved or to be solved as efficiently as possible, require the execution of concurrent actions. In this article we specifically study temporally-expressive problems, in other words, problems that cannot be solved without using the concurrency of actions. A typical example of this class of problem is cooking: several ingredients or dishes must be cooked simultaneously in order to be ready at the same moment. As another example, driving a car requires concurrent actions on the steering wheel and the pedals. Large-scale applications include the management of an airport or a railway station. In industrial environments, concurrency of actions is often essential to keep storage space and turn-around times within given limits.

The majority of temporal planners are incapable of solving temporally expressive problems [6]. Apart from HTN-type planners such as IxTeT [9], [13] or HSTS [19], only CRIKEY3 [5], VHPOP [29], LPGP [14], TLP-GP [16] [17], TM-LPSAT [24], [11] and STEP [12] can solve this type of problem.

Unfortunately, some of these planners are, in fact, incomplete since they are not guaranteed to find a solution to problems involving a cyclically-dependent set of actions (temporally-cyclic problems). A simple example of this type of problem is the construction of two pieces of software, written by two different subcontractors, each needing to know the specification of the other program in order to correctly build the interface between the two programs. In fact, cyclic dependency is surprisingly commonplace. For example, a bank will lend me money provided I pay it back after some mutually-agreed number years, and I will pay it back provided the bank lends me the money in the first place. Many contracts (such as between an employer and an employee or between two companies) can be seen as a mutually-agreed plan to solve temporally-cyclic problems. For example, a condition for an employee to start work is that they will be paid at the end of the month, and a condition for the employer to pay their salary is that the employee did indeed work during the given month.

This article is structured as follows. After reviewing previous work in this field (Section 2) and giving a formal description of the problem studied in this paper (Section 3), we characterize the temporal sublanguages of PDDL 2.1 (Planning Domain Description Language) [15], [8] which can express temporally-cyclic problems (Section 4). We then propose a polynomial-time transformation which converts temporally-cyclic problems into equivalent acyclic problems (Section 5). Applying this transformation restores the completeness of temporally-expressive planners. We finish by summarising and discussing possible avenues of future research.

II. RELATED WORK

Temporally-expressive planners are essentially based on three different types of algorithm [18]:

State-space search: Cushing et al. describe TEMPO [6], a temporally-expressive algorithm based on a complete search in an extended state-space in which decisions concerning when to execute an action are made after all decisions concerning which actions to execute have been made (lifting over time). The CRIKEY3 planner [10], [5], 2008) performs classical

1530-1311/10 $26.00 © 2010 IEEE
DOI 10.1109/TIME.2010.19

IEEE computer society

planning and, in case of failure, switches to a TEMPO-type algorithm.

Partial-order planners: partial-order planners (POP) have been successfully extended to the temporally-expressive framework with planners such as VHPOP [29] and DT-POP [23]. Many other planners have in the past used a hierarchical plan-space (HTN). They use a temporal logic based on instants and intervals, together with a Time Map Manager which manages the temporal constraints. This is the case for planners such as FORBIN [7], HSTS [19], IXTET [9], [13], TEST [20], TIMELOGIC [2], TLP [27], and TRIPTIC [22]. Although their representation languages are very expressive, there are many differences with PDDL 2.1, which makes the comparison with other systems particularly difficult. The limited experimental comparisons which have been performed highlighted their relatively poor performance.

Temporal extensions of GRAPHPLAN: The use of the planning graph [3] has also been extended to temporally-expressive problems. In LPGP [14] and TM-LPSAT [24] durative actions are decomposed into three instantaneous actions (start, invariant, end). Both of these planners use a solver to extract a solution. However, whereas LPGP uses backward search in the planning graph while maintaining the consistency of a set of temporal constraints, TM-LPSAT simultaneously codes the planning graph and the temporal constraints then calls the LPSAT solver [28]. [11] also describes a similar method: all actions are decomposed into two simple actions, then the problem is coded as a CSP according to a semantics based on modal operators and which includes the coding of the planning graph. The TLP-GP planner [16] [17] uses similar methods to those of LPGP and TM-LPSAT constructing a smaller search space. It also delegates a larger part of search to a disjunctive temporal problem solver [25] which probably explains why it is able to outperform LPGP, VHPOP2.2 and CRIKEY3.

III. Incompleteness of planners on temporally-cyclic problems

A temporal planning problem consists of a set A of actions $a = \langle Cond(a), Eff(a), Duration(a)\rangle$, an initial state I and a goal G. I and G are sets of propositions. Initially only the propositions in I are true and, after the execution of all the actions in the plan, all propositions in the goal state G must be true. For each action a: Cond(a) is a set of propositions p together with an instant (relative to the start of a) at which p is required to be true; Eff(a) is a set of propositions p together with an instant (relative to the start of a) at which p is produced. A temporal plan is a set of instances of actions $\langle a, t_{start}\rangle$, where a is an action and t_{start} its corresponding start time. For each action a in the temporal plan, each of its conditions $p \in Cond(a)$ must be true at the instant when it is required by a, and each of its effects $p \in Eff(a)$ is assigned true at the instant when it is produced by a. More precisely, when a variable $p \in Eff(a)$ is assigned a value by a at time t, this assignment has effect immediately *after* t, i.e. on an open interval starting at time t. The value of p is considered undefined for other actions (and other instances of a) at the exact instant t. However, we make the reasonable assumption that it is possible for the same instance of an action to simultaneously test the value of a variable p and to modify it.

In the following sections, we restrict ourselves to the PDDL 2.1 language and its sublanguages (described in detail in Section 4). If we allow several instances of the same action to overlap, then testing the existence of a valid plan becomes an EXPSPACE-complete problem [21]. This is why in this article, as in almost all previous work in temporal planning, we do not authorize two instances of the same action to execute in parallel. This means that we conserve the PSPACE-complete complexity of classical planning [4]. Indeed, we also impose the slightly stronger condition that no two instances of the same action touch (i.e. there is always some non-empty interval between the end of one instance and the start of the following instance). One way to impose this non-overlapping/non-touching constraint on instances of the same action is by introducing a propositional variable (a token) f_a for each action a: f_a is initially true (i.e. $f_a \in I$), f_a is added as a condition and $\neg f_a$ as an effect at the start of a, and f_a is added as an effect at the end of a.

We restrict ourselves to planners which are capable of solving temporally-expressive problems [6]. We start by showing that certain of these planners (LPGP, TLP-GP) are incomplete since they are incapable of finding a solution to a certain type of problem which we now define. Consider the simple example of the problem of interfacing two programs, described in Section 1. A temporal plan for this simple temporally-expressive problem is given in Figure 1. Each rectangle represents a durative action $a = \langle Cond(a), Eff(a), Duration(a)\rangle$. The conditions Cond(a) are shown above the action a and the effects Eff(a) below. The duration of the action is given in square brackets. In this problem, the initial state is empty, the goal consists in building the two programs to be interfaced (effects r and s) and:

- action A (of duration 4) corresponds to the construction of the first program for which we need to know, in order to complete it, the specifications of the second program (condition p).
- action B (of duration 2) corresponds to the construction in parallel of the second program for which we need to know, in order to

complete it, the specifications of the first program (condition q).

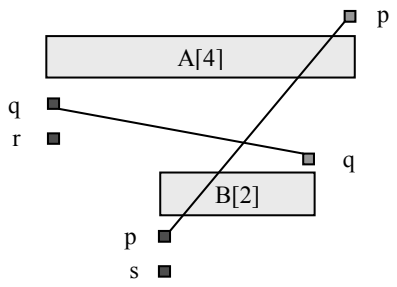

Figure 1. Temporal plan for the program-interfacing problem.

This temporal plan is characteristic of a class of plans that we call *temporally-cyclic* and that can be formalised as follows:

Definition 1 (cyclic set of actions): A set of actions $Q = \{a_1, \ldots, a_n\}$ is a cyclic set if there is a set of propositions $\{p_1, \ldots, p_n\}$ such that $\forall i \in \{1,\ldots, n-1\}$ $p_i \in Eff(a_i) \cap Cond(a_{i+1})$ and $p_n \in Eff(a_n) \cap Cond(a_1)$. We denote this cyclic set together with the propositions p_i by: $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$.

Definition 2 (temporally-cyclic plan): A temporal plan is temporally-cyclic if it contains a set of action-instances $\{<a_1,t_1>,...,<a_n,t_n>\}$, where $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \ldots \rightarrow p_n \rightarrow a_1$ is a cyclic set of actions, and is such that deleting p_i from $Eff(a_i)$ in the action-instance $<a_i,t_i>$ (for any $i \in \{1,\ldots, n\}$) would invalidate the plan (i.e. $<a_i,t_i>$ must produce p_i in order to satisfy the conditions of $<a_{i+1},t_{i+1}>$ (for each $i \in \{1,...,n-1\}$) and $<a_n,t_n>$ must produce p_n for a_1).

It is easily verified that the temporal plan of Figure 1 is temporally-cyclic. Some state-of-the-art temporally-expressive planners, such as TLP-GP, are incapable of producing the temporal plan shown in Figure 1. A planner will not be able to solve this problem if search for a solution or the construction of the planning graph is performed forwards starting from the initial state (and actions are only triggered if their preconditions can be established by already triggered actions).

For this particular problem, in a simple forward search or during the construction of the planning graph starting from the initial state, it is impossible to produce $r \in G$ or $s \in G$ because neither of the conditions {p, q} required by the actions {A, B} can be produced from I. Indeed, neither of the two actions A or B can be triggered.

In a backward search (such as used by VHPOP), this difficulty does not apply since, as the choice of actions is based on the goals, actions A and B will necessarily be selected. However, this is not sufficient to guarantee the completeness of the search algorithm. Indeed, if the heuristic which guides the choice of actions does not prefer those actions which are already present in the plan, then the algorithm can enter an infinite loop. In our example problem, the algorithm could insert a new instance of action A to produce the condition q required by each instance of B (instead of using the instance of A already present in the plan) and similarly insert a new instance of action B to produce the condition p required by each instance of A.

IV. TEMPORAL EXPRESSIVITY AND CYCLIC SETS OF ACTIONS

[6] use the notation $L_{effets}^{conditions}$ to represent the expressivity of a temporal-planning language, where <conditions> and <effects> represent the instants at which the conditions must be true and those at which the effects are produced. The values taken by <conditions> and <effects> can be:

- s: the start of the action;
- e: the end of the action;
- o: (only concerns conditions) over all the duration of the action.

For example, the notation $L_{s,e}^{o}$ denotes a language for which the conditions of each action must be true over all its duration, and each of its effects can occur either at the start instant or the end instant of the action. PDDL2.1 is the temporally-expressive language $L_{s,e}^{s,o,e}$. [6] also give a necessary and sufficient condition for a domain of planning problems to be temporally expressive. This allows them to partition the space of temporal languages into:

- Temporally-expressive languages in which it is possible to represent problems whose solution requires concurrency of certain actions.
- Temporally-simple languages in which it is not possible to represent such problems.

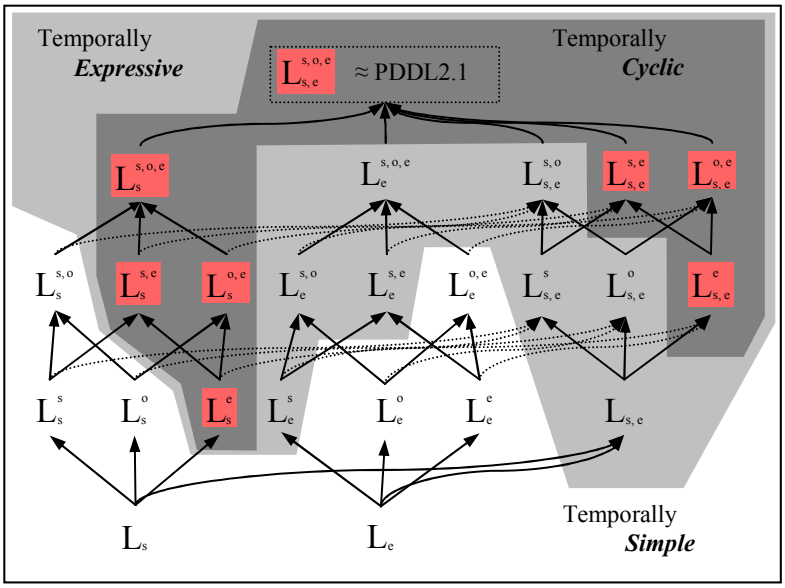

Figure 2. Taxonomy of temporal languages and their expressivity.

They also give the following definitions and prove the temporal-expressivity theorem which we give below.

Definition 3 (before-condition/after-condition):

- A *before-condition* is a condition which is required at the same time as, or before, the production of at least one of the effects of the action.
- An *after-condition* is a condition which is required at the same time as, or after, the production of at least one of the effects of the action.

Definition 4 (temporal gap):

- A gap between two disjoint temporal intervals which do not meet (in the sense of the primitive "meets" of [1]) is the interval between them (and such that the union of these three intervals forms a single interval).
- An action has a *temporal gap* if there is a gap between a before-condition and an effect, or between an effect and an after-condition, or between two effects.

Theorem 1 [6]: A sub-language of PDDL2.1 is temporally simple if and only if it disallows temporal gaps.

We will now show that a certain type of temporal gap allows a language to express temporally-cyclic problems.

Notation: Let a be an action, with $p \in Cond(a)$ and $e \in Eff(a)$. In a temporal plan containing an action-instance $\langle a,t \rangle$, we use $\tau(p \rightarrow \langle a,t \rangle)$, or more simply $\tau(p \rightarrow a)$ when no confusion is possible, to represent the instant at which this instance of a requires the condition p and $\tau(\langle a,t \rangle \rightarrow e)$, or more simply $\tau(a \rightarrow e)$, to represent the instant at which this instance of a produces e.

Definition 5 (temporally cyclic/acyclic language): A language is temporally cyclic if it authorises the existence of temporally-cyclic plans. Otherwise, it is temporally acyclic.

Theorem 2: A sub-language of PDDL2.1 is temporally cyclic if and only if it authorises temporal gaps between an effect and an after-condition.

This theorem, whose proof is given below, allows us to refine the taxonomy of temporal languages proposed by [6] by distinguishing a subclass of temporally-expressive languages corresponding to temporally-cyclic languages. This new taxonomy is given in Figure 2.

Proof of Theorem 2: (1) Firstly, we show that if a sub-language of PDDL2.1 authorises temporal gaps between an effect and an after-condition then it authorises temporally-cyclic plans. The problem whose solution is given in Figure 1 (consisting of the two actions A and B) demonstrates that the language L_s^e authorises the existence of temporally-cyclic plans. As all sub-languages of PDDL2.1 which authorise temporal gaps between an effect and an after-condition

contain L_s^e as a sub-language [6], all these languages authorise cyclic sets of actions.

(2) Secondly, we show that if a sub-language of PDDL2.1 authorises the existence of temporally-cyclic plans, then it authorises temporal gaps between an effect and an after-condition. Consider a temporal planning problem with a temporally-cyclic plan. This means that the temporal plan contains a set of action-instances $\{<a_1,t_1>,...,<a_n,t_n>\}$, where $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \ldots \rightarrow p_n \rightarrow a_1$ is a cyclic set of actions (i.e. $\forall i \in \{1,\ldots, n-1\}\ p_i \in (Eff(a_i) \cap Cond(a_{i+1})) \wedge (p_n \in Eff(a_n) \cap Cond(a_1))$). Furthermore, $<a_i,t_i>$ must produce p_i to satisfy the conditions of $<a_{i+1},t_{i+1}>$ (for each $i \in \{1,...,n-1\}$) and $<a_n,t_n>$ must produce p_n for a_1. It follows that $(\forall i \in \{1,\ldots, n-1\}\ \tau(a_i \rightarrow p_i) < \tau(p_i \rightarrow a_{i+1})) \wedge (\tau(a_n \rightarrow p_n) < \tau(p_n \rightarrow a_1))$. Suppose that there cannot be a temporal gap between an effect and an after-condition of an action. Then $(\forall i \in \{1,\ldots, n-1\}\ \tau(p_i \rightarrow a_{i+1}) \leq \tau(a_{i+1} \rightarrow p_{i+1})) \wedge (\tau(p_n \rightarrow a_1) \leq \tau(a_1 \rightarrow p_1))$. Therefore $\tau(a_1 \rightarrow p_1) < \tau(p_1 \rightarrow a_2) \leq \tau(a_2 \rightarrow p_2) < \tau(p_2 \rightarrow a_3) \leq \ldots \leq \tau(a_n \rightarrow p_n) < \tau(p_n \rightarrow a_1) \leq \tau(a_1 \rightarrow p_1)$. From which we can deduce that $\tau(a_1 \rightarrow p_1) < \tau(a_1 \rightarrow p_1)$, which is impossible. Therefore there is a temporal gap between an effect and an after-condition.

An important question is whether it is possible to model temporally-cyclic problems differently so as to avoid temporal cycles. The next section shows that it is indeed possible to reduce $L_{s,e}^{s,o,e}$ (i.e. the whole of PDDL2.1) to the temporally-acyclic language $L_{s,e}^{s,o}$ by modelling differently actions which have a temporal gap between an effect and an after-condition.

V. Transformation of a Temporally-Cyclic Problem into a Temporally-Acyclic Problem

In this section we show how to transform a problem whose solutions potentially contain cycles, expressed in the PDDL 2.1 language $L_{s,e}^{s,o,e}$, into an equivalent problem expressed in the language $L_{s,e}^{s,o}$ which by Theorem 2 does not authorise cycles. It follows that we can use forward search to solve all temporally-expressive problems. In practice, according to the problem, we could transform all actions, which amounts to expressing the problem in the language $L_{s,e}^{s,o}$, or only transform those actions which can possibly lead to the creation of cycles, while still remaining in the language $L_{s,e}^{s,o,e}$. The proof of Theorem 2 shows that a necessary condition for a problem to have a temporally-cyclic solution-plan is the existence of a cyclic set of actions $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$ involving a temporal gap between an effect p_i and an after-condition p_{i-1} of some action a_i.

In order to detect such actions, we construct the causality graph <V,E> where the set of vertices V is the union of the set of actions and the set of propositions, and the set of directed edges $E = \{<p,a>: p \in Cond(a)\} \cup \{<a,p>: p \in Eff(a)\}$. For each candidate action A (i.e. an action which has a gap between an effect and some after-condition p), we check a necessary condition for the existence of a cycle: does the link $p \rightarrow A$ belong to a cycle in the causality graph. If not, then there is no need to transform A.

An action A which has after-conditions which could cause a cycle must be transformed into an action A' for which these conditions have been deleted (to eliminate the temporal gap). Each such deleted after-condition p of A is reformulated as a link between A and the presence in the plan of some action B which produces this condition p for A. (In fact B is the last, in the temporal order of the plan, to produce p before it is needed by A). This link must be active whenever A is present in the plan. The presence of this link is checked by an action $LC_{p,A}$ (Link Check between A and the action B which produced p for A).

To transform an action A in order to delete an after-condition p, we:

- Add a proposition Linked(p,A) to the initial state and to the goal of the problem. This link-proposition will mean that, when A is executed and deletes Linked(p,A) (see below) this forces the execution of the check action $LC_{p,A}$ to re-establish Linked(p,A). $LC_{p,A}$ can only execute if some action B actually produced p for A.
- Transform A into A':
  - by deleting the after-condition p,
  - by adding the proposition LC-Active(p,A) at the instant when p was required, to denote the fact that p must be produced by some action before it is required by A,
  - by adding the effect ¬Linked(p,A) which forces the execution of the action $LC_{p,A}$ to re-establish Linked(p,A) (this proposition being present in the initial state and the goal),
  - by adding the effect ¬LC-Active(p,A) at the start of A' to ensure that $LC_{p,A}$ is executed after the end of A'.

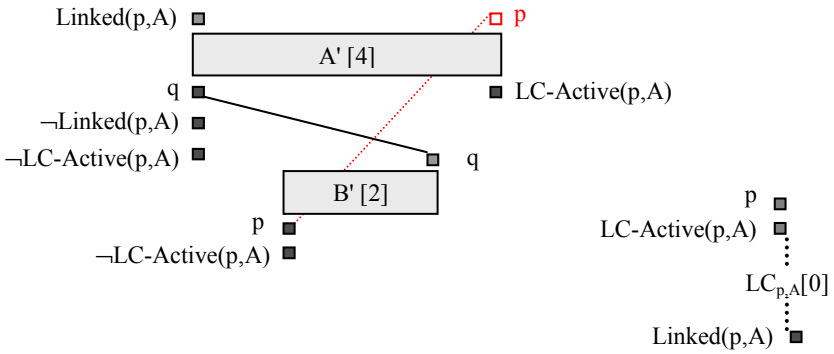

Figure 3. Example of the transformation of a temporally-cyclic problem into a temporally-acyclic problem.

- For each action B which produces p, transform B into B' by adding the effect ¬LC-Active(p,A) at the instant when B produces p to denote the fact that p must be produced (by at least one of these actions B) *before* being required by A.
- Add an instantaneous action $LC_{p,A}$ (Link Check) with the two conditions p and LC-Active(p,A) and with the effect Linked(p,A).
- In the special case that A also destroys p at the end of A, we also need to perform the following two modifications:
  - the effect ¬p is deleted from the end of A' and added to $LC_{p,A}$,
  - for each action C (apart from A) such that p∈Cond(C), transform C into C' by adding the effect ¬LC-Active(p,A) at the instant when p is required by C, to ensure that such actions cannot be executed between the end of A' and $LC_{p,A}$.

These new propositions and actions cannot introduce new temporal gaps between effects and after-conditions since all the conditions which are added during the transformation are added at the start of actions. The resulting problem is equivalent to the original problem and the inverse transformation of a temporal plan is achieved by simply deleting the actions $LC_{p,A}$. We prove formally the equivalence of the original and transformed problems in the Appendix.

In our running example, for action A with after-condition p, the transformation is illustrated in Figure 3.

The complete transformation of the problem into the temporally acyclic language $L^{s,o}_{s,e}$ can be achieved via the systematic deletion of all after-conditions (without testing for cycles in the causality graph) via the transformation described above. Let n denote the size of the problem, i.e. n = Σ(|Eff(a)|+|Cond(a)|) where the sum is over all actions a (assuming, without loss of generality, that each action has at least one condition or effect). Then this simple algorithm has a complexity of $O(n^2)$. This is because, for each action A with an after-condition p∈Eff(A), we may need to transform all other actions B. The transformation of the actions A and B, for a given after-condition p∈Eff(A), requires constant time.

```
Transformation Algorithm
Candidates←{actions with effect/after-condition gap}
While Candidates ≠ ∅
  Extract A from Candidates ;
  For each after-condition p of A which causes a gap
    Build the causality graph G from p → A ;
    If p → A belongs to a cycle in G then
      Transform-Cond-and-Eff(A,p,A') ;
      For each action B which produces p
        Transform-Eff1(B,p,B') ;
      End For ;
      If A simultaneously requires and destroys p
      then
        For each action C ≠ A which requires p
          Transform-Eff2(C,p,C') ;
        End For ;
      End If ;
      Add the action LC_p,A to the problem ;
    End If ;
  End For ;
End While ;
```

Figure 4. Transformation algorithm using the causality graph.

The worst-case time complexity of the partial transformation of a problem, in which we eliminate only those after-conditions that could potentially give rise to cycles (detected by causality graph analysis) is also $O(n^2)$, but this algorithm has the advantage that the

number of new actions introduced will generally be less. The algorithm is given in Figure 4. For each action A and each after-condition $p \in Eff(A)$, we firstly test whether there is a cyclic set of actions including $p \rightarrow A$. This can be achieved by finding the strongly-connected components of the causality graph G using the algorithm of [26]. On a directed graph <V,E>, this algorithm has a time complexity of O(|V| + |E|). But this is O(n), since n is exactly equal to the number of directed edges. A complete transformation can therefore be achieved in $O(n^2)$ by the argument above. In both cases, the number of new actions $LC_{p,A}$ (as well as the number of new propositions LC-Active(p,A) and Linked(p,A)) is bounded above by the number of after-conditions present in the actions of the problem. The size of the transformed problem is therefore linear compared to the size n of the original problem.

VI. Conclusion and Future Work

In this article we have shown that certain temporally-expressive planners are not complete for all sublanguages of PDDL2.1. In particular, they are incapable of solving problems containing sets of cyclically-dependent actions. We have given a method which, thanks to a polynomial-time transformation of problems containing such cycles, allows these planners to solve problems containing sets of cyclically-dependent actions. This method could be applied to future temporally-expressive planners.

Possible avenues of future research opened up by the work presented in this article include the completeness of algorithms which use even more expressive languages. We are currently working on the extension of our present results to planning problems whose actions are defined using arbitrary temporal intervals. In the long term, one can also imagine the extension to numerical planning involving incertitude and continuous transitions.

VII. Appendix

Let TEMPORAL PLANNING($\mathcal{L}$) represent the set of instances of the temporal planning problem belonging to the language $\mathcal{L}$.

Theorem 3: There is a polynomial-time reduction from TEMPORAL PLANNING($L_{s,e}^{s,o,e}$) to TEMPORAL PLANNING($L_{s,e}^{s,o}$).

Proof of Theorem 3: It suffices to show that the transformation given in Section 5 produces an equivalent problem, since we have already shown in Section 5 that it is a polynomial reduction. Let Π and Π' be the original and transformed problems, respectively.

Let P be a valid plan for Π, and let P' be identical to P except that all actions have been transformed (i.e. A becomes A', B becomes B', etc.). Let δ be strictly positive but smaller than the interval between any two distinct times at which a condition or effect occurs during the execution of the plan P. For each instance of A' in P', we add an instance of $LC_{p,A}$ at a time δ after the end of A'. Since nothing happens during this interval of length δ after the end of A', by definition of δ, the combined effect of this instance of A' and the following instance of $LC_{p,A}$ is equivalent to A in the original plan P.

Now consider a plan P' for Π'. Let P be obtained from P' by deleting all instances of $LC_{p,A}$ and by replacing all transformed actions by their original versions (i.e. A' by A, B' by B, etc.). We will show that P is a valid plan for Π. The condition LC-Active(p,A) of $LC_{p,A}$ is only established by action A' (at its end). Furthermore, LC-Active is destroyed at the start of A' and remains false during the execution of A', since, by assumption, no two instances of A' can overlap. Therefore, no instance of $LC_{p,A}$ can occur during the execution of A'. The condition Linked(p,A) of A' is only established by $LC_{p,A}$. Furthermore, A' destroys Linked(p,A) which is one of the goals of Π'. It follows that every instance of A' must be followed by an instance of $LC_{p,A}$ in P'. We must show that p is true at the end of the execution of each instance of A'. We know that p is true when the following instance of $LC_{p,A}$ is executed, so we only have to show that p was not established between the end of A' and the execution of $LC_{p,A}$. But this is impossible since all actions B' which establish p also destroy LC-Active(p,A). In the case that the effect $\neg p$ is transferred from the end of A to $LC_{p,A}$, we have to show that no action C' in P' uses p as a condition during the interval between the end of A' and the execution of $LC_{p,A}$. Again, this is impossible since all such actions C' also destroy LC-Active(p,A).

References

[1] J.F. Allen, "Towards a General Theory of Action and Time", *Artificial Intelligence 23(2)*, pp. 123-154, 1984.

[2] J.F. Allen, J.A.G.M. Koomen, "Planning Using a Temporal World Model", *Proc. of 8th International Joint Conference on Artificial Intelligence*, pp. 741-747, 1983.

[3] A.L. Blum, M.L. Furst, "Fast planning through planning-graphs analysis", *Proc. of 14th International Joint Conference on Artificial Intelligence*, pp. 1636-1642, 1995.

[4] T. Bylander, "The Computational Complexity of Propositional STRIPS Planning", *Artificial Intelligence 69(1-2)*, pp. 165-204, 1994.

[5] A. Coles, M. Fox, D. Long, A. Smith, "Planning with Problems Requiring Temporal Coordination", *Proc. of AAAI 2008*, pp. 892-897, 2008.

[6] W. Cushing, S. Kambhampati, Mausam, D.S. Weld, "When is Temporal Planning Really Temporal?", *Proc. of 20th International Joint Conference on Artificial Intelligence*, pp. 1852-1859, 2007.

[7] T. Dean, J. Firby, D. Miller, “Hierarchical Planning involving deadlines, travel time and resources”, *Computational Intelligence 6(1)*, pp. 381-398, 1988.

[8] M. Fox, D. Long, “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains”, In *Journal of Artificial Intelligence Research 20*, p. 61-124, 2003.

[9] M. Ghallab, A.M. Alaoui, “Managing Efficiently Temporal Relations Through Indexed Spanning Trees”, *Proc. of 11th International Joint Conference on Artificial Intelligence*, pp. 1297-1303, 1989.

[10] K. Halsey, D. Long, M. Fox, “CRIKEY - A Planner Looking at the Integration of Scheduling and Planning”, *Proc. of the “Integration Scheduling Into Planning” Workshop*, *ICAPS'03*, pp. 46-52, 2004.

[11] Y. Hu, “Temporally-Expressive Planning as Constraint Satisfaction Problems”, *Proc. of 20th International Joint Conference on Artificial Intelligence*, pp. 192-199, 2007.

[12] R. Huang, Y. Chen, W. Zhang, “An Optimal Temporally Expressive Planner: Initial Results and Application to P2P Network Optimization”, *Proc. of 19th International Conference on Automated Planning and Scheduling*, 2009.

[13] P. Laborie, M. Ghallab, “Planning with Sharable Resource Constraints”, *Proc. of 14th International Joint Conference on Artificial Intelligence*, pp. 1643-1651, 1995.

[14] D. Long, M. Fox, “Exploiting a graphplan framework in temporal planning”, *Proc. of 13th International Conference on Automated Planning and Scheduling*, pp. 52-61, 2003.

[15] D.V. McDermott, “PDDL, The Planning Domain Definition Language”, In *Technical Report*. http://cs-www.cs.yale.edu/homes/dvm/, 1998.

[16] F. Maris, P. Régnier, “TLP-GP: Solving Temporally-Expressive Planning Problems”, *Proc. of 15th International Symposium on Temporal Representation and Reasoning*, pp. 137-144, 2008.

[17] F. Maris, P. Régnier, “TLP-GP: New Results on Temporally-Expressive Planning Benchmarks”, *Proc. of 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 507-514, 2008.

[18] F. Maris, P. Régnier, “Planification temporellement expressive”, *Revue d'Intelligence Artificielle*, n° spécial "Planification, Décision et Apprentissage", to appear, 2010.

[19] N. Muscettola, “HSTS: integrating planning and scheduling”. In M. Zweben & M. Fox Eds. *Intelligent Scheduling*, pp. 169-212, 1994.

[20] Reichgelt H. & Shadbolt N. (1990). A Specification Tool for Planning Systems, *Proc. of 9th European Conference on Artificial Intelligence*, pp. 541-546.

[21] J. Rintanen, “Complexity of Concurrent Temporal Planning”, *Proc. of 17th International Conference on Automated Planning and Scheduling*, pp. 280-287, 2007.

[22] E. Rutten, J. Hertzberg, “Temporal Planner = Nonlinear Planner + Time Map Manager”, *Artificial Intelligence Communications 6(1)*, pp. 18-26, 1993.

[23] P. Schwartz, M.E. Pollack, “Planning with Disjunctive Temporal Constraints”, *Proc. of ICAPS'04 Workshop on Integrating Planning into Scheduling*, 2004.

[24] J. Shin, E. Davis, “Continuous Time in a SAT-Based Planner”, *Proc. of 19th National Conference on Artificial Intelligence (AAAI'04)*, pp. 531-536, 2004.

[25] K. Stergiou, M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints", *Artificial Intelligence 120(1)*, pp. 81-117, 2000.

[26] R. Tarjan, “Depth-first search and linear graph algorithms”, *SIAM Journal on Computing 1(2)*, pp. 146-160, 1972.

[27] E. Tsang, “TLP – a temporal planner”, *on Advances in artificial intelligence*, John Wiley & Sons, pp. 63-78, 1987.

[28] S. Wolfman, D. Weld, “The LPSAT Engine and its Application to Resource Planning”, *Proc. of 16th International Joint Conference on Artificial Intelligence*, pp. 310-317, 1999.

[29] H.L.S. Younes, R.G. Simmons, “VHPOP: Versatile Heuristic Partial Order Planner”, *Journal of Artificial Intelligence Research*, 20, pp. 405-430, 2003.