

On the Compilation of Plan Constraints and Preferences

Stefan Edelkamp

Computer Science Department,
University of Dortmund, Germany
stefan.edelkamp@cs.uni-dortmund.de

Abstract

State trajectory and preference constraints are the two language features introduced in PDDL3 (Gerevini & Long 2005) for describing benchmarks of the 5th international planning competition. In this work we make existing solver technology applicable to planning with PDDL3 domains by compiling the new constructs back to PDDL2 (Fox & Long 2003).

State trajectory constraints are translated into LTL formulae and further to Büchi automata, one for each constraint. These automata are compiled to grounded PDDL and the results are merged with the grounded representation of the original problem. *Preference constraints* are compiled away using numerical state variables. We provide encouraging experimental results in heuristic search planning.

State Trajectory Constraints

State trajectory constraints provide an important step of the agreed fragment of PDDL towards the description of *temporal control knowledge* (Bacchus & Kabanza 2000) and *temporally extended goals* (DeGiacomo & Vardi 1999). In the proposed translation, we compile state trajectory constraints to LTL formulae and Büchi automata (Clarke, Grumberg, & Peled 2000).

All plans are finite, so that we can cast the Büchi automaton as an NFA, which accepts a word if it *terminates* in a final state. The labels of the NFA are conditions over the propositions and variables in a given state. For example, the constraint *a fragile block can never have something above it* in the Blocksworld domain is expressed as

```
(always (forall (?b - block)
  (implies (fragile ?b) (clear ?b)))
```

The instantiated LTL formula for two blocks *a* and *b* is

```
[] ((fragile_a -> clear_a) &&
  (fragile_b -> clear_b))
```

The corresponding automaton is shown in Fig. 1. The automaton consists of only one state.

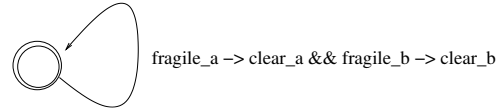


Figure 1: Automaton with a transition in clause form.

Numeric conditions, e.g. (always (<= (fuel-used) 10)), do not lead to additional expressiveness as for the translation process they are temporarily viewed as propositions for constructing the transition labels. When applying such an automata, the construction has to be followed by a backward translation of edges to numeric conditions.

The assertion *each block should be put on the table at least once* corresponds to

```
(forall (?b - block) (sometime (ontable ?b)))
```

and is called an *every/sometime* constraint. For two blocks a Büchi automaton with respect to the LTL formula (<> ontable_a) && (<> ontable_b) is constructed. It is shown in Fig. 2 (left).

The statement *in some state visited by the plan all blocks are on the table* is expressed as

```
(sometime (forall (?b - block) (ontable ?b)))
```

denoted as *sometime/every* constraint. The according LTL formula is <> (ontable_a && ontable_b) with a Büchi automata shown in Fig. 2 (right).

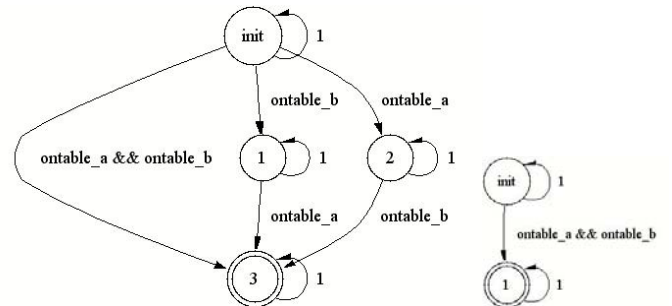


Figure 2: Büchi automata for the *every/sometime* constraint (left) and *sometime/every* constraint (right).

Encoding

To encode the synchronized simulation of the automata, we devise a predicate (`at ?n - state ?a - automata`) to be instantiated for each automata state and each automata that has been computed. For detecting accepting states, we utilize instantiations of the predicate (`accepting ?a - automata`). The initial state includes the start state of the automata and an additional proposition if it is accepting. For all automata, the goal specification includes automata acceptance.

Next, we have to specify allowed automata transitions in form of planning actions by declaring a grounded operator for each automata transition, with the current automaton state and the transition label as preconditions, as well as the current automaton state as delete and the successor state as add effects. As state trajectory constraints are fully instantiated, adding transition labels to operator preconditions can only be achieved in the grounded problem representation. For transitions leading to an accepting state we include the corresponding automata acceptance proposition to the add effect list. As we require a tight synchronization between the constraint automaton transitions and the operators in the original planning space, we include *synchronization flags* that are flipped when an ordinary or a constraint automaton transition is chosen. An example for a grounded automata transition (in Blocksworld) is the following propositional operator

```
(:action sync-trans-a-0-init-a-0-accept-0
:precondition (and (at-a-0-init)
 (sync-automaton-a-0) (on-a-b))
:effect (and (accepting-a-0)
 (not (at-a-0-init)) (at-a-0-accept-0)
 (not (sync-automaton-a-0)) (sync-ordinary)))
```

As said, the size of the Büchi automaton for a given formula can be exponential in the length of the formula¹.

Metric Time Constraints

So far we have seen how to derive and compile Büchi automata for the simple plan constraints `sometime` and `always`. (Gerevini & Long 2005) show that `at-most-once`, `sometime-before` and `sometime-after` can also be expressed using standard LTL expressions, such that these modalities easily fit into the above framework. For *metric time constraints* like `within`, `always-within`, `hold-during`, and `hold-after` we have to restrict actions to the execution time window specified in the constraints. Moreover, these constraints necessarily call for parallel/temporal planning, as they refer to absolute points in time for plan execution.

These expressions are tackled using *timed initial literals* (TILs) as already contained in the language

¹In the notion of (Nebel 2000) the compilation is *essential*. Similar essential compilations have been proposed by (Gazen & Knoblock 1997) for ADL to STRIPS and by (Thiebaux, Hoffmann, & Nebel 2005) for derived predicates.

PDDL2 (Hoffmann & Edelkamp 2005). TILs denote fixed dates in plan time at which an atom is set to *true* or to *false*. As they are allowed to be checked only in the operators' preconditions, TILs correspond to *action execution time windows*. The modalities `hold-after` and `hold-during` immediately translate to TILs for the operators, in which the stated conditions are satisfied in the preconditions. If the state formula is disjunctive the planner has to deal with multiple action windows.

For combined metric and temporal modalities like `within` and `always-within`, action execution time window are included in form of additional timed initial literal for the preconditions of the automata's transitions. The modality (`always-within t p q`) is included into the inferred automata according to the conversion rule (Gerevini & Long 2005): `always p → (within t q)`.

Preferences

Preferences model soft constraints that are desirable but do not have to be fulfilled in a valid plan. The degree of desirability of satisfying a preference constraints is specified in the plan metric.

Simple preferences refer to ordinary propositions (and are included to the planning goal). E.g., if we prefer block *a* to reside on the table during the plan execution, we write (`preference p (on-table a)`) with a validity check (`is-violated p`) in the plan objective. Such checks are interpreted as natural numbers that can be scaled and combined with other variable assignments in the plan metric. To evaluate these cost for a given plan, we have to accumulate how often the stated preference condition is violated in the preconditions of the actions in the plan. The according numerical value is substituted in the metric for its evaluation. Quantified preference constraints like

```
(forall (?b - block) (preference p (clear ?b)))
```

are flattened to multiple instantiated preference conditions (one for each block), while the inverse expression (`preference p (forall (?b - block) (clear ?b))`) leads to only one constraint.

Preferences for state trajectory constraints like (`preference clean`

```
(forall (?b - block) (always (clean ?b))))
```

can, in principle, also be dealt with automata theory. Instead of requiring to reach an accepting state we *prefer* to be there, by means that not arriving at an accepting state incurs costs to the evaluation of the plan metric using the (`is-violated clean`) variable.

There is one subtle problem. As trajectory constraints may prune the exploration, violating it can also be due to a failed transition in the automata. E.g., the constraint automata of Fig. 1 prunes the exploration on each operator in which the only transition is not satisfied. Our solution is to have an extra operator (for each automata) that allows to skip the enforced synchronization. Applying this operator is assigned to corresponding costs and move the automata to a dead(-end) state.

Encoding

For each atomic preferences p we include a numerical variable `is-violated-p` in the grounded domain description. If the preferred predicate is contained in the *delete list* of the operator then the variable is set to 1, if it is contained in the *add list*, then the variable is set to 0, otherwise it remains unchanged. For complex propositional or numerical goal conditions ϕ in a preference condition p , we use *conditional effects* (Gazen & Knoblock 1997) as follows:

```
(when ( $\neg\phi$ ) (assign (is-violated-p) 1))
(when ( $\phi$ ) (assign (is-violated-p) 0))
```

For preferences p on trajectory constraints, variable `(is-violated-a-p)` annotates the automata states in an obvious way. If the automata accepts, the preference is fulfilled, so the value of `(is-violated-a-p)` is set to 0. If it enters a non-accepting state it is set to 1. The `skip` operator also induces setting the variable to 1.

Experimental Results

We wrote a *3-phase* compiler, which translates PDDL3 into grounded PDDL2. In the *first phase* for each state trajectory constraint we parse its specification, flatten the quantifiers and write the corresponding LTL-formula to disk. The *second phase* derives a Büchi-automaton for each LTL formula and generates the corresponding PDDL code. This conversion, called *ltl2pddl* is derived from extending the source code of the existing compiler *LTL2BA*². The *third phase* merges the results of the first and second phase. We apply heuristic search forward chaining search, i.e., ordinary *FF* (Hoffmann & Nebel 2001) for propositional domains, *Metric-FF* (Hoffmann 2003) for domains with numerical variables. Note that by translating plan preference, otherwise propositional problems are compiled into metric ones. For temporal domains, we extended *Metric-FF* to handle temporal operators and timed initial literals. The resulting planner can deal with disjunctive action time windows and uses an internal linear-time scheduler to derive parallel plans (Edelkamp 2003).

We draw a sample set of experiments on an Intel Pentium IV, 3 GHz, 2 GB Linux PC. Time ($\leq 1/2$ h) is measured in CPU seconds. It includes the parsing time of the grounded and merged representations but not the time for generating the translated input.

Table 1 shows the result of FF on the compiled result of a *sometime/every* property in *DriverLog* (Strips, IPC-3). We compare the plan lengths and the exploration efficiencies neglecting the constraints with the ones of the planner including the constraints using either *enforced hill climbing* (EHC) or *best-first search* (BF). As expected, the sequence of operators in compiled plans alternate between original plan space actions and automata transitions so that their length is even.

The plan length generally increases³. Although the

²www.liafa.jussieu.fr/~oddoux/ltl2ba.

³The artifact in the first problem is due to a shorter plan not searched by EHC, which becomes attracted with the

	without stc		EHC with stc		BF with stc	
	length	time	length	time	states	trans
1	8	0.00	2 × 7	0.00	2 × 7	0.00
2	23	0.01	2 × 25	0.04	2 × 24	0.01
3	12	0.00	2 × 16	0.01	2 × 17	0.01
4	20	0.01	2 × 26	0.01	2 × 25	0.04
5	23	0.01	2 × 27	1.09	2 × 27	0.09
6	13	0.11	2 × 33	237.61	2 × 33	27.05
7	10	0.01	2 × 25	6.92	2 × 25	0.77
8	29	0.03	2 × 44	334.14	2 × 33	23.29
9	33	0.05	-	-	2 × 33	5.67
10	15	0.01	-	-	2 × 30	4.54

Table 1: *DriverLog* with the state trajectory constraint (`sometime (forall (p - obj) (at ?p s0))`).

	without stc		EHC with stc		BF with stc	
	length	time	length	time	states	trans
1	8	0.00	2 × 7	0.00	2 × 7	0.00
2	23	0.01	2 × 24	0.04	2 × 24	0.01
3	12	0.00	2 × 16	0.02	2 × 17	0.01
4	20	0.01	2 × 24	0.04	2 × 24	0.04
5	23	0.01	2 × 33	0.46	2 × 27	0.09
6	13	0.11	2 × 35	3.46	2 × 37	0.48
7	10	0.01	2 × 26	0.50	2 × 24	0.21
8	29	0.03	2 × 47	13.62	2 × 33	2.35
9	33	0.05	2 × 57	3.69	2 × 48	0.43
10	15	0.01	2 × 40	2.12	2 × 36	0.63

Table 2: *DriverLog* with the state trajectory constraint (`forall (p - obj) (sometime (at ?p s0))`).

planning time for the unconstrained case is shorter as the constrained one, the length of the plans for BF are often slightly better than the ones for EHC. This also seems natural, as EHC is supposed to be less conservative. EHC failed to find plans in larger problems, while BF produce solutions in an adequate time and space. The reason lies the directed planning state-space problem graph. Even if the original problem graph is undirected, the intersection with the Büchi automaton makes the it directed.

In Table 2 we show the impact of an *every/sometime* constraint. In this case EHC scales much better, and both algorithm obtain a better exploration result both in plan length but especially in exploration time. This was to be expected as the constraint of having an object at a place for some point in time is less restrictive as requiring all objects to be there at some point in time. The sizes of the automata grow exponentially, such that the size of the automata is not a good indicator of the difficulty for solving the problem.

In Table 3 we show the effect of combining constraints in *ZenoTravel* (IPC-3). Different to the unified automata in Fig. 2, we have used the cross-product of two individual automata for expressing *sometimes* and *at-*

imposed constraint.

	without stc		EHC with stc		BF with stc	
	length	time	length	time	states	trans
1	1	0.00	3 × 2	0.01	3 × 2	0.01
2	8	0.00	3 × 10	0.01	3 × 6	0.01
3	6	0.00	3 × 6	0.03	3 × 6	0.03
4	13	0.01	3 × 10	0.04	3 × 11	0.03
5	11	0.00	3 × 11	0.05	3 × 11	0.04
6	13	0.01	3 × 16	0.07	3 × 13	0.06
7	16	0.01	3 × 16	0.06	3 × 16	0.07
8	13	0.02	3 × 13	0.18	3 × 15	0.29
9	25	0.05	3 × 22	0.21	3 × 24	0.45
10	25	0.05	3 × 28	0.30	3 × 28	0.38

Table 3: *ZenoTravel* domain with the state trajectory constraint (and (at-most-once (at plane1 city1)) (sometime (at plane1 city1))).

	BF with pc			BFBnB with pc		
	length	time	quality	length	time	quality
1	10	0.06	2,103	10	0.04	1,784
2	22	0.36	2,078	22	1.06	2,078
3	15	0.09	1,663	15	0.29	1,387
4	18	0.24	2,122	18	3.41	2,122
5	19	1.45	1,747	21	18.17	1,745
6	11	0.22	2,369	11	15.91	1,843
7	17	13.67	3,038	17	96.01	2,502
8	23	1,570	2,153	23	1,570	2,153
9	23	0.68	2,901	21	4.76	2,901
10	21	1.39	2,770	21	7.48	2,394

Table 4: *DriverLog* Domain (Numeric) with the preference constraints (preference p1 (at truck1 s0)), (preference p2 (at truck2 s1)), and (preference p3 (at truck3 s2)).

most-once. Consequently we see that the lengths of the synchronized plans are a multiple of three. The variation of the lengths produced by EHC and BF is small, favoring one or the other. Since EHC has a lower overhead on the maintenance of states its plan finding times are smaller. Compared to FF, we do not observe a large plan length offset especially for larger domains.

For preferences constraints we chose *Metric-FF* to apply best-first search. We added branch-and-bound on top of it to improve the solution quality after the first plan has been found. The relaxed plan length is used as the heuristic. In *DriverLog* (Numeric) the expression

```
(+ (+ (* 500 (is-violated-p1)) (* 400
(is-violated-p2))) (* 300 (is-violated-p1)))
```

has been added to the plan metric. The results are shown in Table 4. Optimization pays-off in most cases.

Conclusion

We have seen a Büchi automaton interpretation for state-trajectory constraints and considered a metric interpretation of preference constraints. An effective translation of the new language fragment PDDL3 to

PDDL2 has been provided that allows already existing planners to deal with the new expressiveness. Regardless of the promising experiments with heuristic search planners, the approach is planner-independent.

It is, however, not difficult to internalize the proposed methods. The biggest challenges for planning with constraints in future besides the development of search heuristics are refined optimization methods to arrive at good solutions fast. Additional to various local search approaches, we plan to evaluate other best-first search algorithms compatible with branch-and-bound, like *beam* and *beam-stack search* (Zhou & Hansen 2005).

Acknowledgements The author thanks DFG for continuous support in the projects ED 74/2 and Ed 74/3, and the IPC-5 committee for intense discussion.

References

- Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:123–191.
- Clarke, E.; Grumberg, O.; and Peled, D. 2000. *Model Checking*. MIT Press.
- DeGiacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, 226–238.
- Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.
- Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.
- Gazen, B. C., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *ECP*, 221–233.
- Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.
- Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research* 24:519–579.
- Hoffmann, J., and Nebel, B. 2001. Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Hoffmann, J. 2003. The Metric FF planning system: Translating “Ignoring the delete list” to numerical state variables. *Journal of Artificial Intelligence Research* 20:291–341.
- Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.
- Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.
- Zhou, R., and Hansen, E. 2005. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, 90–98.