Planning in Dynamic Worlds: More Than External Events

Alexander Nareyek and Tuomas Sandholm

Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891, USA
`alex@ai-center.com`, `sandholm@cs.cmu.edu`

Keywords: planning, multi-agent systems

Abstract

Hardly any environment is a static domain in which an agent is the only one who is changing states of the world. Thus, many events occur that are not a direct consequence of an agent's actions. Besides events that occur in full independence of an agent, there are events that can be influenced indirectly, e.g., by asking other agents to perform specific tasks. Even in the absence of other agents, an action by an agent can lead to a complex effect chain. Explicitly reasoning about such indirect consequences of actions is indispensable in nearly all real-world domains.

Action planning systems to drive an intelligent agent, however, do not incorporate concepts to handle such world dynamics. An action usually has a definite effect — without a possibility of further indirect consequences. Some planning systems allow for the occurrence of external events, but do not enable relations to an agent's actions.

In this paper, we propose a solution to this shortcoming of planning systems by integrating a kind of enhanced rule-based system. Using this approach, a planning system can reason about indirect consequences and exploit these external mechanics in order to achieve its goals. The enhancements are demonstrated from the perspective of a constraint-based planning system based on local search.

1 Introduction

Producing behavior commands for an autonomous agent can be achieved by a large variety of methods. Pre-coded behavior scripts are one of the most applied and extreme alternatives, providing highly domain-adequate guidance, but quickly become infeasible in slightly more complex environments. When more sophisticated AI techniques are to be applied, the question shifts to the issue which feature set can efficiently be handled. For example, BDI logics [11] provide a very expressive framework, but involve unmanageable complexity. Approaches of action planning, which serve as a basis in the following, provide a middle ground. They focus on assembling/scheduling networks of basic action primitives instead of allowing for arbitrary reasoning problems, thereby making it possible to apply highly specialized technology to speed up the solving process.

The basic planning problem is given by a partial description of an agent's current world, a partial description of the goal world, and a set of action/operator types that map a partial world description to another partial world description. A solution is a sequence of action instances transforming the current world description to the goal world description and is called a *plan*.[1] The problem can be enriched by including further aspects, like temporal or uncertainty issues, or by desiring the optimization of certain properties.

Action planning systems, however, still lack many features that would be needed for appropriately guiding an autonomous agent. One of them — agent-external changes of the world/environment — is dealt with in this paper. We should mention that we do not target a distributed planning domain; every agent is fully autonomous and interacts with his environment only to maximize his personal goal achievement.

Action planning systems are very agent-centric, i.e., hardly ever involve any other mechanisms/agents acting in the world. In very rare cases, *external events* and transitions among them are considered (e.g., in [1]). These events, however, cannot be influenced by the agent's actions. In most cases, such events are used to model the incomplete knowledge of an agent. Examples of contingency planners, i.e., planning systems that construct branching plans to deal with the uncertain event outcomes, are Warren's WARPLAN-C [10], CNLP by Peot and Smith [8], Plinth by Goldman and Boddy [5] and Cassandra by Pryor and Collins [9]. There are probabilistic extensions as well, including Drummond and Bresina's synthetic projection [4] and the BURIDAN probabilistic planning by Kushmerick, Hanks and Weld [6] with its contingent extension by Draper, Hanks and Weld [3]. Recently, much research in this area has focused on planning based on Markov decision processes (see [2] for an overview). But again, these approaches do not involve the influence of the agent, and hardly ever model world mechanics that produce these events.

---

[1] Note that this definition of the term *planning* is different from that expected by people in the operations research (OR) community (e.g., *scheduling*).

Planning in a multi-agent world requires more than handling external events. The agent needs to know about the mechanics at work — *external transitions* —, so that he can influence and potentially exploit these mechanisms. For example, an agent could order another agent to follow him. This action does not have a direct post-condition that the follower will always be at the same position as the agent but only that the follower will try to do so. The agent needs to take care that the follower does not lose track of him, e.g., when the follower has to stop at a red traffic light. It is impossible to solve such a scenario with a conventional planning system because such a system cannot reason about the external transitions and the impact of the command to follow. In theory, it would be possible to incorporate all possible external changes as a by-product of temporally very fine-grained actions of the agent — this, however, involves an unmanageable combinatorial explosion for any realistic application.

The differentiation between actions and external transitions may look strange to persons from the planning community at first, but the key point is that the world's behavior can be modeled as a set of ongoing transitions/rules instead of final, static consequences.

External transitions can be handled by introducing a kind of rule-based system. However, it is not sufficient to fire all applicable rules until a fix point is reached. The agent wants to exploit external transitions, e.g., to have another agent performing a task for him, and must be able to chain backwards through these rules to determine how the changes can be initiated. Reasoning about such rules is in principle similar to reasoning about the agent's own actions; however, there are some important differences. The following section introduces our underlying planning model, which will be used in the following to demonstrate the techniques' realization.

2 The Planning Model

Describing the complete model and the underlying concepts of the Excalibur agent's planning model [7] would go beyond the limits of this paper. Only the basic elements are thus described in this section, and the following sections will provide simplified intuitive descriptions as well as some more in-depth information for those that are familiar with the system.

2.1 Problem Specification and Solution

Planning is specified in an extended constraint-programming framework. Basic elements are *variables* and *constraints*. Constraints limit the possible values that the variables can take. Because of our local-search approach, variables have concrete values at any time during the search process. Cost functions are specified for each constraint type, returning a value that represents the constraint's current inconsistency with respect to the connected variables. For example, a simple `Sum` constraint with two variables a and b to be added and an s variable for the sum could specify its costs as $\texttt{Sum}_{costs} = |a + b - s|$. If the sum of all constraints' costs reaches zero, the variables' value assignments represent a valid solution to the problem.

For the generation of a solution, a constraint has a number of heuristics to improve its cost function. For example, a heuristic for the `Sum` constraint could randomly choose one of the related variables and change its value such that the constraint is fulfilled. Another heuristic might resolve the inconsistency by distributing the necessary change such that all variables are changed by the same (minimal) amount. A constraint must make the choice as to which heuristic to apply on its own. In each improvement iteration of local search, one of the constraints that has costs higher than zero is selected to perform an improvement.

2.2 Model Elements

Figure 1 shows an example constraint graph involving the constraints that are of interest in the following. The constraint graph represents a very simple plan — involving only one action to move from location A to B. *Object constraints* are a structural feature to group graph elements, i.e., they do not restrict variables' values and do not have a cost function.

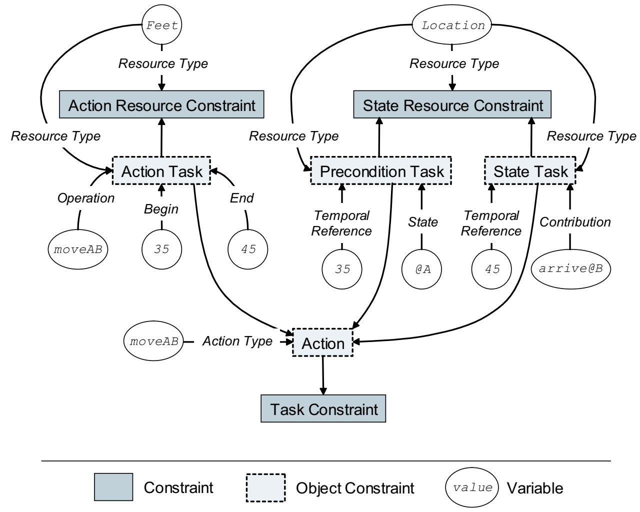


Figure 1: An example constraint graph for planning.

An Action (e.g., moving from A to B) consists of a set of different preconditions that must be satisfied (e.g., that the agent is at location A), operations that must be performed (e.g., the actual low-level commands to execute the movement) and resulting state changes (e.g., the agent's location being changed to B). These elements are represented by object constraints, i.e., there are Precondition Tasks for precondition tests, Action Tasks for operations and State Tasks for state changes. There are three basic types of regular constraints:

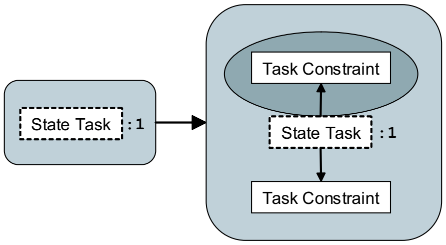
- An Action Resource Constraint (ARC) checks if there is enough capacity to carry out the operations, i.e., that the connected tasks do not overlap (e.g., that the feet do not move to multiple locations at once).
- A State Resource Constraint (SRC) checks if the connected Precondition Tasks are satisfied by the states that are deduced from the temporal projection of the connected State Tasks (e.g., that the "@A" precondition of the action to move from A to B is met).

- A Task Constraint (TC) represents action requirements by specifying a relation between a set of Precondition Tasks, Action Tasks and/or State Tasks (connected via the Action object constraint). The constraint is satisfied if the tasks represent the action type given by the action-type variable, i.e., if the correct tasks are involved and if the tasks fulfill certain restrictions (e.g., that the Precondition Task and the Action Task of the movement action begin at the same time).

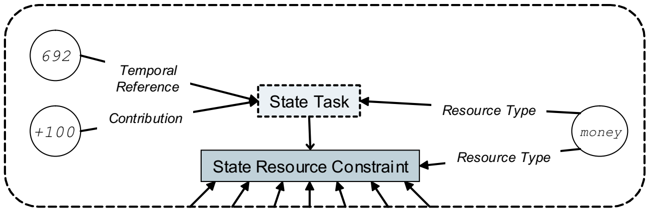
2.3 Graph Structure

To solve planning problems, it is not sufficient to change variables' values. The constraint graph itself must be changed. For example, it must be possible to add/remove actions. Thus, the constraints' heuristics can also execute graph modifications, e.g., to add the structures for an extra state task.

So-called *structural constraints* ensure that graph modifications of the heuristics do not produce an invalid structure of the constraint graph. Figure 2 shows one of the structural constraints for the planning domain.



An example of a structural constraint. The constraint applies at any part of the graph where the left side matches, and is fulfilled if the right side matches then as well (the dark area of the right side describes a "negative" match, i.e., exactly one task constraint is required to be connected to the state task).



A partial view of a structurally inconsistent constraint graph. The structural constraint above matches the state task but does not find a connected task constraint.

Figure 2: An example of a structural constraint.

Similar to regular constraints, structural constraints have cost functions to express their satisfaction and heuristics for improvement iterations.

3 External Transitions

As described above, each action of an agent explicitly lists its effects on the world (in our approach, by way of State Tasks). The problem is that there are often many indirect consequences. All possible indirect consequences of an action could of course be integrated in an action by conditional effects, but the combinatorial explosion involved renders this approach infeasible. Imagine enumerating all possible external consequences on future situations of an action to say "Yes" instead of only modeling the direct effect that the agent's consent is expressed. Indeed, the basic idea of planning techniques is *not* to pre-code all possible situations.

Changes of the world that are beyond the agent's direct control are called *external transitions* is the following. In respect to incorporating them into the planning process, external transitions can in general be modeled similar to the regular actions of an agent. Nevertheless, there are some important differences:

- **Occurence:** External transitions occur beyond the agent's control, i.e., they definitely occur, and the agent has no choice of adding them to the plan or not (they can only be influenced indirectly).
- **Temporal placement:** An external transition occurs *exactly* at the time when all of its preconditions start to be satisfied. The planning system cannot shift it to *some* point in time when the preconditions are satisfied.

4 An Example Problem

Our agent — Little Red Riding Hood — wants to leave grandmother's house. In addition, she wants the wolf to stay outside the house so that he cannot eat her grandmother:

```
goal:
 own.location(t in [0..horizon]) =
   outside
 wolf.location([0..horizon]) = outside
```

She is currently in the living room with her grandmother, while the wolf is somewhere outside:

```
facts:
 own.location(0) = living_room
 grandmother.location(0) = living_room
 grandmother.goal(0) = idle
 wolf.location(0) = outside
 door.passage(0) = locked
```

To get outside, Red Riding Hood needs to walk to the entrance room, unlock the door, and walk outside. She cannot lock the door from outside but can tell her grandmother to lock the door behind her:

```
action walk_to_entrance:
 pre: own.location(t) = living_room
 eff: own.location(t+10) = entrance_room

action unlock_door:
 pre: own.location(t) = entrance_room
      door.passage(t) = locked
 eff: door.passage(t+1) = unlocked

action walk_outside:
 pre: own.location(t) = entrance_room
      door.passage(t) = unlocked
 eff: own.location(t+1) = outside
```

```
action request_locking:
 pre: own.location(t) = living_room
      grandmother.location(t) =
        living_room
 eff: grandmother.goal(t+2) = lock_door
```

Besides Red Riding Hood's own actions, there are additional external transitions. The grandmother will do as requested, and the wolf will get in if the door is unlocked for more than a second:

```
transition grandmother_walks_to_entrance:
 pre: grandmother.goal(t) = lock_door
      grandmother.location(t) =
        living_room
 eff: grandmother.location(t+20) =
        entrance_room

transition grandmother_locks_door:
 pre: grandmother.goal(t) = lock_door
      grandmother.location(t) =
        entrance_room
      door.passage(t) = unlocked
 eff: door.passage(t+1) = locked

transition wolf_enters_house:
 pre: wolf.location(t) = outside
      door.passage([t,t+1]) = unlocked
 eff: wolf.location(t+2) = entrance_room
```

Figure 3 shows an overview of the actions and transitions of this domain.

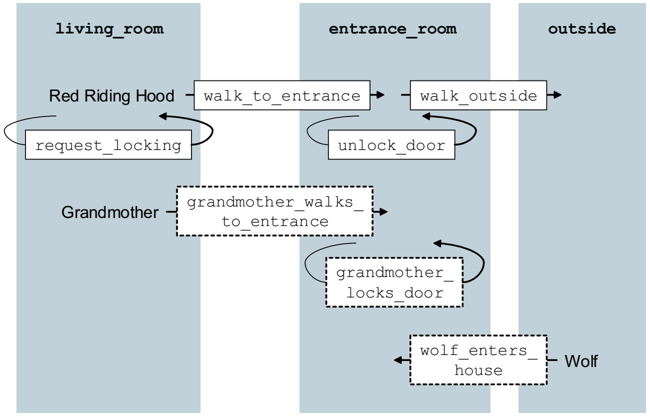


Figure 3: Actions and transitions of the Little Red Riding Hood example.

5 Firing External Transitions

During the regular planning process, the world states need to be scanned for matching preconditions of external transitions, so that the corresponding effects can be inserted. It is similar to applying a rule-based system (extended by temporal annotations) until a fix point is reached. External transitions can be realized by the regular structures of ACTIONS/PRECONDITION TASKS/STATE TASKS (no ACTION TASKS of course), allowing for additional values for action-type variables.

The problem for our local-search approach is, however, how the addition ("firing") of all applicable external transitions can be enforced. To do so, a new constraint — the TRANSITION CONSTRAINT (TrC) — is introduced. When a TrC is introduced in the graph, it notifies all SRCs of the graph that they should keep the TrC informed about all intervals at which states hold that are tested by the external transitions' preconditions. If a new SRC is introduced during the solving process, it also searches the graph for TrCs and asks them for states to be monitored for them.

Using the notifications of the SRCs, the TrC internally monitors the development of the external transitions' precondition states to see if and when external transitions need to fire (see Figure 4). In the example, each transition fires once, but of course, the transitions may generally fire multiple times over time.

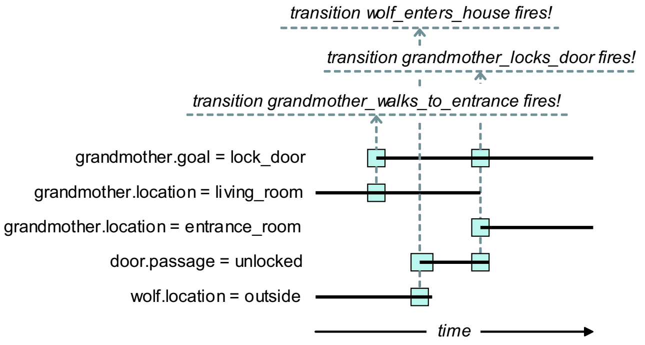

transition wolf_enters_house fires!
transition grandmother_locks_door fires!
transition grandmother_walks_to_entrance fires!
grandmother.goal = lock_door
grandmother.location = living_room
grandmother.location = entrance_room
door.passage = unlocked
wolf.location = outside
time

Figure 4: Keeping track of relevant states inside the TRANSITION CONSTRAINT.

The TrC can be connected to an ACTION object constraint, indicating that the corresponding action is an external transition instead of an action executed by the agent. The TrC involves costs if an external transition needs to fire, but a corresponding ACTION is not connected. Similarly, it involves costs if an ACTION is connected for which the firing conditions of an external transition are not fulfilled (see also Section 6.1). The TRANSITION CONSTRAINT's main improvement heuristics simply add or remove ACTIONs.

Figure 5 shows the corresponding graph specification parts for the TRANSITION CONSTRAINT.

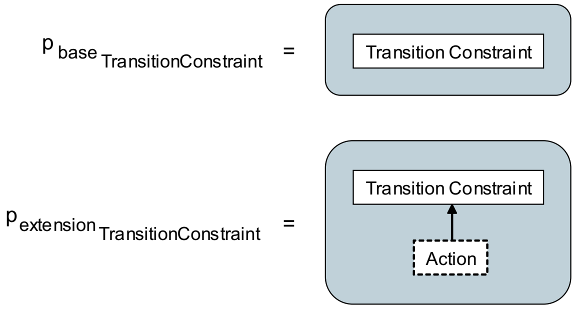
Usually, only one TrC should exist in the graph because they might do redundant work otherwise. The structural constraint shown in Figure 6 ensures that there is exactly one TRANSITION CONSTRAINT in the constraint graph at a time.

6 Planning with External Transitions

We want to exploit the functionality of external transitions, not only passively incorporating their effects. This means that it must be possible to actively plan external transitions.

6.1 Desired External Transitions

An SRC's heuristic can add a desired external transition to change a specific state just like a regular action would be



$p_{base\,TransitionConstraint}$ =
Transition Constraint
$p_{extension\,TransitionConstraint}$ =
Transition Constraint
Action

Figure 5: Graph definition of the TRANSITION CONSTRAINT.

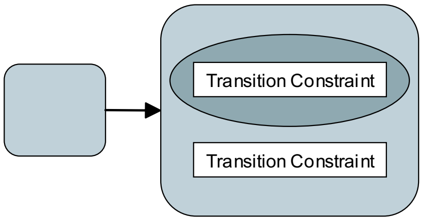


Figure 6: The structural constraint $S_{exactlyOneTrC}$.

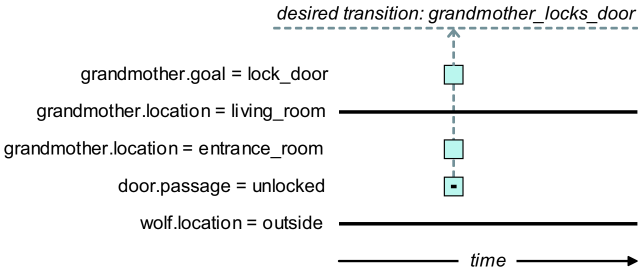
added. The desired external transition will not be valid at that time because otherwise, it would have been inserted by the TrC before. The plan needs to be repaired so that the desired transition becomes a "real" external transition.

In case of a regular action, the enforcement of the correct temporal placement of a precondition can be pursued in a more isolated way. For an external transition, there is an additional condition: At least one of the preconditions must be placed at the time when its tested state has just changed. This "at least one" makes checking and enforcing the consistency of a desired external transition a very global issue. However, because of our modular constraint-programming approach, the information when a state change occurs is only available within an SRC's local state projection.

Luckily, the introduced TRANSITION CONSTRAINT already has the global information on the relevant state changes available because the SRCs notify it about them. Repairing the firing-related consistency of desired external transition should thus be a task for the TrC (see Figure 7). To guarantee that a TrC will note that an ACTION representing a desired external transition was added, an extra cost-function component is added to the TASK CONSTRAINT that requires an ACTION with an action-type variable representing an external transition to be connected to the TrC.

Assigning the repair of a desired external transition to a TrC is accomplished by adding a cost-function component to it that represents the firing-related consistency of all connected external transitions, i.e., the condition that at least one precondition is placed at a state change. For the repair heuristics, however, a new concept must be introduced in our solver: *improvement delegation*.

The repair of the firing-related consistency can be done in multiple ways. The simplest one is a shift of a precondition task, which could be done by the TrC itself. But there are others possible repairs that require the knowledge of an SRC. For example, instead of moving the precondition itself, its preceding state change could be moved — which means moving the state tasks that realize the state change. Because of this, a TrC can delegate the improvement to one of the preconditions' SRCs, which in turn have specialized heuristics to deal with this request.



desired transition: grandmother_locks_door
grandmother.goal = lock_door
grandmother.location = living_room
grandmother.location = entrance_room
door.passage = unlocked
wolf.location = outside
time

Figure 7: A desired but not yet consistent external transition.

The delegation of an improvement is not needed in general, but the more sophisticated knowledge available at another constraint may speed up the solving process enormously. Note that these efficiency gains have a downside: The initiating constraint requires the existence of the constraint to which the improvement is to be delegated. This opposes constraint programming's concept of an arbitrary composition of the modular problem components (i.e., constraints and variables). A heuristic that wants to delegate an improvement should thus test the existence of the other constraint before, and provide an alternative repair if the other constraint is not part of the constraint graph. In addition, a constraint needs to keep track which improvements have been delegated during an iteration in order to prevent delegation cycles.

6.2 Manipulating Existing External Transitions

Besides enforcing desired external transitions, existing external transitions will often be challenged because an SRC wants to undo or move the related effect. For example, in the planning situation of Figure 4, the SRC responsible for projecting the wolf's location and having to ensure the goal that the wolf is outside, will try to remove the state task of the transition `wolf_enters_house`. If this transition is a consistent one instead of a desired, this removal might however be useless because the TrC will most likely re-establish the transition.

Again, the delegation concept can be applied. The SRC will hand the improvement over to the TrC, which has the necessary overview to initiate appropriate changes. In many cases, the TrC will again delegate the improvement to an SRC, e.g., to invalidate the precondition that the door is unlocked for a longer time. Nevertheless, infinite calling cycles cannot occur because the delegations will always move strictly backward with respect to the temporal projection (the current time being a final stop).

7 Conclusion

The ability to handle external transitions was based on a kind of underlying rule-based system, whose firing transitions describe the world's dynamics. The planning system uses the knowledge of these transitions to plan for actions that change the world in an indirect way, e.g., by making another agent to execute a service.

Using a modular search framework, such as constraint programming, implies many challenges, mostly related to the global nature of the firing conditions of external transitions. In our approach, we have thus introduced a specific constraint that centrally collects the necessary information. Realizing a plan-improvement step is then often interplay between the local constraints and the central one, leading to the concept of improvement delegation.

The ability to handle and exploit external transitions is an important step in pushing the functionality of action planning to a level at which it can be useful for guiding autonomous agents in dynamic environments and multi-agent domains. Many open questions still remain and will be tackled in our future work. For example, firing all possible transitions may not be computationally feasible in environments that are more complex. Trade-offs between the amount of knowledge about the environment and the time to compute it must be explored, and it is important to develop priority schemes which knowledge to extract.

More information on the underlying EXCALIBUR project and a sample implementation are available on the project's website:
`http://www.ai-center.com/projects/excalibur/`

References

[1] Blythe, J. Planning with External Events. In Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'94), 94–101, 1994.

[2] Boutilier, C.; Dean, T.; and Hanks, S. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11: 1–94, 1999.

[3] Draper, D.; Hanks, S.; and Weld, D. Probabilistic Planning with Information Gathering and Contingent Execution. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 31–36, 1994.

[4] Drummond, M. E., and Bresina, J. L. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), 138–144, 1990.

[5] Goldman, R. P., and Boddy, M. S. Conditional Linear Planning. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 80–85, 1994.

[6] Kushmerick, N.; Hanks, S.; and Weld, D. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76: 239–286, 1995.

[7] Nareyek, A. *Constraint-Based Agents – An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds*. Reading, Springer LNAI 2062, 2001.

[8] Peot, M., and Smith, D. Conditional Nonlinear Planning. In Proceedings of the First International Conference on AI Planning Systems, 189–197, 1992.

[9] Pryor, L., and Collins, G. Planning for Contingencies: A Decision-based Approach. *Journal of Artificial Intelligence Research* 4: 287–339, 1996.

[10] Warren, D. H. D. Generating Conditional Plans and Programs. In Proceedings of the Summer Conference on Artificial Intelligence and Simulation on Behavior, 344–354, 1976.

[11] Wooldridge, M. *Reasoning about Rational Agents*. Reading, MIT Press, 2000.