On the Path from Classical Planning to Arithmetic Constraint Satisfaction

Éric Jacopin*
CREC Saint-Cyr
Écoles de Coëtquidan
56381 GUER Cedex
France
ejacopin@acm.org

Jacques Penon
Laboratoire de Théories Géométriques
Université de Paris 7
2 Place Jussieu
75005 Paris Cedex 05
France

From: AAAI Technical Report WS-00-02. Compilation copyright © 2000, AAAI (www.aaai.org). All rights reserved.

Abstract

We discuss a method to automatically transform a classical planning problem into an arithmetic constraint satisfaction problem (aCSP).

This transformation specifies the variables of the aCSP, their ranges and the binary arithmetic constraints over these variables. A bounds consistency solver is then called to build ranges consistent with the arithmetic constraints.

The transformation presented in this paper only produces totally ordered plans and propositional classical planning.

Introduction

When searching for a solution plan, Partial Order Planning (POP) places a constraint between two operators only, to implement the so-called least commitment approach to precondition establishment and declobbering (CHAPMAN 1987). It is of course possible to introduce several constraint satisfaction problems in this approach: constraining the possible values of operator variables (codesignation constraints (CHAPMAN 1987)), constraining the choice of establishers (multi-contributors (KAMBHAMPATI 1992)), and the building of the partial order itself. All these CSPs have variables whose domain is made of symbolic values; that is, the constraints are not arithmetic constraints whose variables have numerical domains.

We here present an arithmetic model of classical planning. This model is automatically built from the classical planning problem data: the initial state, the final state and the operators of the classical planning problem. The construction is not precondition-based but is establisher/clobberer based which can be seen as a generalization of the so-called causal-link (MCALLESTER & ROSENBLITT 1991).

The paper is organized as follows. We begin with the minimal problem the procedure has to solve to show any interest. We then present what are the variables of the problem and their ranges. We then show how to generate arithmetic constraints from classical planning operators and discuss how to choose the relevant operators for the solving of the classical planning problem. Finally we give the planning procedure.

The reader is assumed to be familiar with both classical partial order planning, e.g., (KAMBHAMPATI, KNOBLOCK, & YANG 1995) and constraint satisfaction techniques, e.g., (MARRIOTT & STUCKEY 1998).

An Arithmetic Model

The variables A classical planning problem can be modelled as an arithmetic binary CSP using the formulas from (i) the initial state, (ii) the final state and (iii) the operators.

As an illustration for the rest of the paper, let us solve the following bottom-up problem:

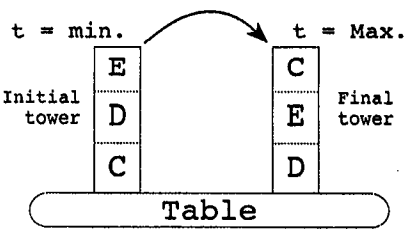

Figure 1: The simplest blocks world problem to be solved with bounds consistency focusing on formulas: one formula is true in the initial state and true in the final state, but false somewhere in-between. The problem is to transform the initial tower into the final tower.

The solution is shown as a plan in Figure 2, using the operators given in (CHAPMAN 1987). Henceforth we refer to this problem as the *bottom-up* problem.

Although a 3 blocks problem, the bottom-up problem is more complex than the classical sussman anomaly because there exists a formula, namely `on(E,D)`, which is true in the initial state, then false and then true again when building the final tower from the initial tower. To represent the evolution of the truth of `on(E,D)`, we need 2 ranges: 1 from the initial state to the state immediately after where block E is on the Table while D

*Thanks to Philippe Morignot for his comments on previous drafts of this paper, to Alexander Nareyek for his help and to Fred Garcia and Pierre Regnier for enlightening discussions.

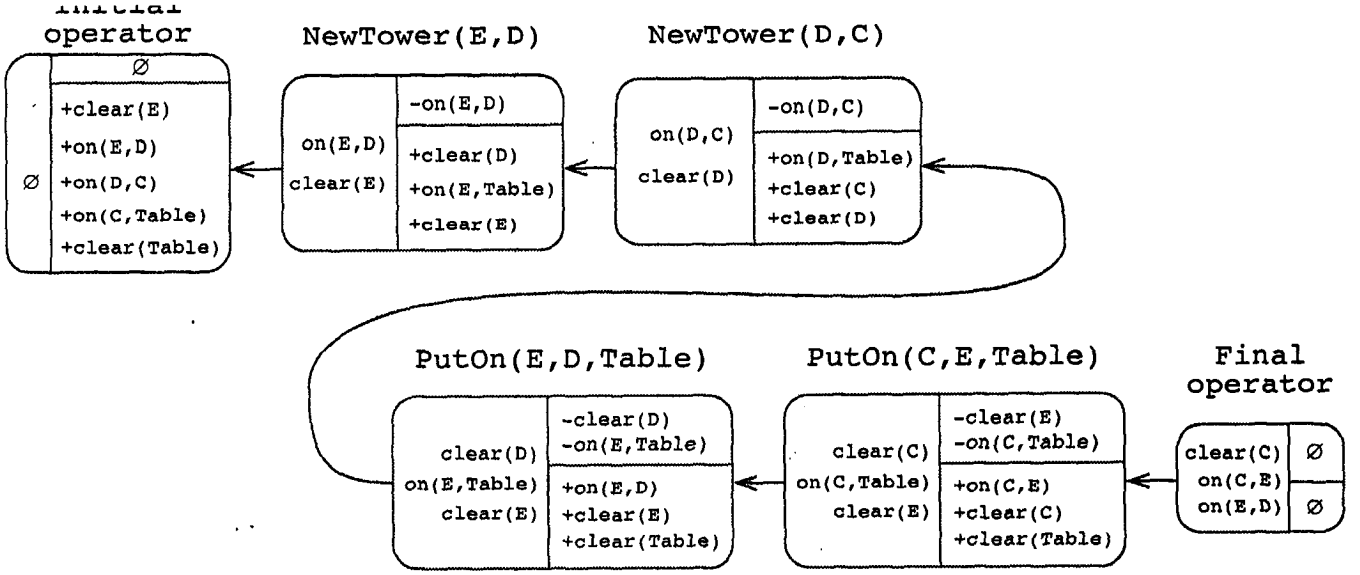

Figure 2: A solution plan, totally ordered partial plan in the space of partial plans; an operator is a round-corner rectangle divided in 3 parts: (i) the left part contains the preconditions, (ii) the upper right part contains the deleted formulas and (iii) the lower right part contains the added formulas. An arrow between operators indicates a precedence between operators. The leftmost operator is the initial operator stating the initial state of the problem while the rightmost operator is the final operator stating the final state.

still is atop C, plus 1 from the state where E is on D and C is on the Table, to the final state. Figure 3 illustrate the evolution of the truth of `on(E,D)` and all the other formulas involved in the solving of the bottom-up problem.

We now face the following choice: either (i) abstracting one formula into one variable with multiple ranges-of-truth or else (ii) abstracting one formula into multiple activation-to-truth (of this very formula) with one range.

We pick the second choice.

The *activation* from false to true is obviously related to the membership of a formula to the add list of an operator and the *deactivation* from true to false is related to the membership to the delete list. To use classical planning terminology (CHAPMAN 1987), an establisher activates a formula while a clobberer deactivates a formula.

The *propagation* of the activation until the deactivation of the formula corresponds to the range of the truth of the formula. If we make graphically explicit the propagation of not only the truth but also the falseness of a formula in the plan of Figure 2, we obtain something close to the plan fo Figure 4. The first important difference between the two plans of the two figures is the addition of a set of *virtual* preconditions, which contains to-be-activated formulas and to-be-deactivated formulas although these formulas are not preconditions of the corresponding operator in Figure 2. The second important difference is the addition of formulas, activated by an operator, to the precondition set of this operator. Consequently, anything added or deleted, i.e., activated or deactivated, is a precondition, either virtual or *real* (i.e., indeed a precondition of the corresponding planning operator).

Note that we cannot say that an added formula comes from a virtual precondition, because it can be the case that a previous operator already activated this formula: it is for instance the case of `clear(Table)` in Figure 4.

We call *activation link*, the tuple made of the formula, the establisher and the clobberer, noted $\alpha(\varphi)$ for short. Such an activation link corresponds to an integer range as illustrated by Figure 4 : for instance, $\alpha(\texttt{clear(Table)}, I_O, F_O)$ corresponds to the integer range $[0, 4]$. There are two activation links for the formula `on(E,D)`: $\alpha_1(\texttt{on(E,D)}, I_O, \text{NewTower(E,D)})$ and $\alpha_2(\texttt{on(E,D)}, \text{PutOn(E,D,Table)}, F_O)$ which correspond to the integer range $[0, 0]$ and $[3, 4]$, respectively.

What the plan construction process needs to compute is when a formula is first activated and then deactivated. Variables of the aCSP are the $\min(\alpha(\varphi))$ and $\max(\alpha(\varphi))$ of the integer ranges of all the activation links of all the formulas involved in a plan.

The ranges The differences from Figure 3 and Figure 4 show the transformation of the time values. In Figure 3, the integer time values stamp the situations. The inclusion of the preconditions of an operator into the set of formulas describing a situation give the preconditions the same time value than the situation. The

formulas of the add and delete lists, used to compute the successive situation, get the successive integer time value.

Stamping the initial situation with 0 is trivial. But POP adds operators to a current plan until the solution is found. That is, when POP starts to build a solution, it does not know how many operators shall be necessary to build the solution plan. This does not fit in the aCSP framework where maximum and minimum values are used to compute new values satisfying the constraints of the problem. One must then find a way to compute this very first maximum value. As illustrated in Figure 4, the maximum value is the number of operator of a totally ordered plan.

Consequently, we only have one choice: the constraints must derive from the operators chosen for some reason. When the choice of operators is over, the maximum value is know (the number of operator) and the solving of the constraints can begin. That's how works the planning procedure we present below, but there still are 2 things to discuss: (i) what constraints can be derived from planning operators? That's what we discuss next; and (ii) how can we choose operators without building the plan? That's what we discuss when presenting the planning procedure.

The arithmetic binary constraints Henceforth, M stands for the maximum integer time value, i_s and f_s for the initial and final state, respectively, and I_o and F_o for the initial and final operator, respectively.

Recall that the variables are the minima and the maxima of the activation links. Minima are when the formulas are activated, i.e., established and maxima are when the formulas are deactivated, i.e., clobbered. So we specify constraints between formulas of the add list, delete list and the preconditions.

We give below the specifications of the constraints in plain english; the reader should refer to Table 1 for their axiomatic counterpart.

Let's begin with the easiest part, the constraints from the initial state and final state:

Initial (1) The initial operator activates all the formulas of the initial state,

Initial (2) All the formulas that the initial operator does not activate are activated by another operator,

Final The final operator deactivates all the formulas of the final state.

We now express the constraints derived from a planning operator O=(Pre,Add,Del) (recall that Add(O)$\cap$Del(O)= $\emptyset$):

Deletions

- If no other operator activates its added formulas, then the deactivation of its deleted formulas exactly precedes the activation of its added formulas,
- if an operator does not activates its added formulas, then the deactivation of its deleted formulas follows the activation of its added formulas,

Preconditions (1) The activation of the preconditions that an operator does not delete is strictly before their deactivation,

Preconditions (2) The preconditions that an operator does not delete can be deactivated when the same operator activates its added formulas,

Preconditions (3) The deactivation of the preconditions that an operator does not delete is strictly after the deactivations of its deleted formulas which can be equal to the activation of the precondition.

As an example, let's specify the constraints from the operators involved in the bottom-up problem. The PutOn(X,Y,Z) operator is the following:

$$\begin{cases} \text{Pre(PutOn)} & : & \{\texttt{on(X,Z)}, \texttt{clear(X)}, \texttt{clear(Y)}\} \\ \text{Add(PutOn)} & : & \{\texttt{on(X,Y)}, \texttt{clear(Z)}\} \\ \text{Del(PutOn)} & : & \{\texttt{on(X,Z)}, \texttt{clear(Y)}\} \end{cases}$$

Applying the axioms in Table 1, we obtain the following constraints from the PutOn(X,Y,Z) operator (no constraint generated for $\min(\alpha(\texttt{on(X,Z)}))$ and $\min(\alpha(\texttt{clear(Y)}))$):

$$\begin{array}{l} \max(\alpha(\texttt{on(X,Y)})) > \max(\alpha(\texttt{on(X,Z)})) \\ \max(\alpha(\texttt{on(X,Z)})) \geq \min(\alpha(\texttt{on(X,Y)})) - 1 \\ \max(\alpha(\texttt{on(X,Y)})) > \max(\alpha(\texttt{clear(Y)})) \\ \max(\alpha(\texttt{clear(Y)})) \geq \min(\alpha(\texttt{on(X,Y)})) - 1 \\ \max(\alpha(\texttt{clear(Z)})) > \max(\alpha(\texttt{on(X,Z)})) \\ \max(\alpha(\texttt{on(X,Z)})) \geq \min(\alpha(\texttt{clear(Z)})) - 1 \\ \max(\alpha(\texttt{clear(Z)})) > \max(\alpha(\texttt{clear(Y)})) \\ \max(\alpha(\texttt{clear(Y)})) \geq \min(\alpha(\texttt{clear(Z)})) - 1 \\ \max(\alpha(\texttt{clear(X)})) \geq \min(\alpha(\texttt{on(X,Y)})) \\ \max(\alpha(\texttt{clear(X)})) \geq \min(\alpha(\texttt{clear(Z)})) \\ \max(\alpha(\texttt{clear(X)})) > \max(\alpha(\texttt{clear(Y)})) \\ \max(\alpha(\texttt{clear(Y)})) \geq \min(\alpha(\texttt{clear(X)})) \\ \max(\alpha(\texttt{clear(X)})) > \max(\alpha(\texttt{on(X,Z)})) \\ \max(\alpha(\texttt{on(X,Z)})) \geq \min(\alpha(\texttt{clear(X)})) \end{array}$$

Here are now the NewTower(X,Z) operator and the constraints it generates (no constraint generated for $\min(\alpha(\texttt{on(X,Z)}))$):

$$\begin{cases} \text{Pre(NewTower)} & : & \{\texttt{on(X,Z)}, \texttt{clear(X)}\} \\ \text{Add(NewTower)} & : & \{\texttt{on(X,Table)}, \texttt{clear(Z)}\} \\ \text{Del(NewTower)} & : & \{\texttt{on(X,Z)}\} \end{cases}$$

$$\begin{array}{l} \max(\alpha(\texttt{on(X,Table)})) > \max(\alpha(\texttt{on(X,Z)})) \\ \max(\alpha(\texttt{on(X,Z)})) \geq \min(\alpha(\texttt{on(X,Table)})) - 1 \\ \max(\alpha(\texttt{clear(Y)})) > \max(\alpha(\texttt{on(X,Z)}) \\ \max(\alpha(\texttt{on(X,Z)}) \geq \min(\alpha(\texttt{clear(Y)})) - 1 \\ \max(\alpha(\texttt{clear(X)})) \geq \min(\alpha(\texttt{on(X,Table)})) \\ \max(\alpha(\texttt{clear(X)})) \geq \min(\alpha(\texttt{clear(Y)})) \\ \max(\alpha(\texttt{clear(X)})) \geq \max(\alpha(\texttt{on(X,Z)})) \\ \max(\alpha(\texttt{on(X,Z)})) \geq \min(\alpha(\texttt{clear(X)})) \end{array}$$

In the above constraints, the two disjoint constraints

$$\max(\alpha(\varphi_\delta)) = \min(\alpha(\varphi_\alpha)) - 1$$

Initial (1)	$\forall \varphi \in i_s, \exists \alpha(\varphi) = \alpha(\varphi, i_O, \_)$ with $\min(\alpha(\varphi)) = 0$
Initial (2)	$\forall \varphi \notin i_s, \exists \alpha(\varphi) = \alpha(\varphi, \_, \_)$ with $\min(\alpha(\varphi)) > 0$
Final	$\forall \varphi \in f_s, \exists \alpha(\varphi) = \alpha(\varphi, \_, F_O)$ with $\max(\alpha(\varphi)) = M$
Deletions	$\forall \varphi_\delta \in \text{Del(O)}, \forall \varphi_\alpha \in \text{Add(O)}$, $\exists \alpha(\varphi_\delta) = \alpha(\varphi_\delta, \_, \text{O}), \exists \alpha(\varphi_\alpha) = \alpha(\varphi_\alpha, \text{O}_\alpha, \_)$ $\forall \text{O}_\alpha \ \max(\alpha(\varphi_\alpha)) > \max(\alpha(\varphi_\delta))$ and if $\text{O}_\alpha = \text{O}$ then $\max(\alpha(\varphi_\delta)) = \min(\alpha(\varphi_\alpha)) - 1$ if $\text{O}_\alpha \neq \text{O}$ then $\max(\alpha(\varphi_\delta)) > \min(\alpha(\varphi_\alpha)) - 1$
Preconditions (1)	$\forall \varphi_\pi \in \text{Pre(O)}$ such that $\varphi \notin \text{Del(O)}$, $\exists \alpha(\varphi_\pi) = \alpha(\varphi_\pi, \_, \_)$ with $\max(\alpha(\varphi_\pi)) > \min(\alpha(\varphi_\pi))$
Preconditions (2)	$\forall \varphi_\pi \in \text{Pre(O)}$ such that $\varphi \notin \text{Del(O)}, \forall \varphi_\alpha \in \text{Add(O)}$, $\exists \alpha(\varphi_\pi) = \alpha(\varphi_\pi, \_, \_), \exists \alpha(\varphi_\alpha) = \alpha(\varphi_\alpha, \text{O}_\alpha, \_)$ with $\forall \text{O}_\alpha \max(\alpha(\varphi_\pi)) \geq \min(\alpha(\varphi_\alpha))$
Preconditions (3)	$\forall \varphi_\pi \in \text{Pre(O)}$ such that $\varphi \notin \text{Del(O)}$, $\forall \varphi_\delta \in \text{Del(O)}$, $\exists \alpha(\varphi_\pi) = \alpha(\varphi_\pi, \_, \_), \exists \alpha(\varphi_\delta) = \alpha(\varphi_\delta, \_, \text{O})$ with $\forall \text{O}_\alpha \max(\alpha(\varphi_\pi)) > \max(\alpha(\varphi_\delta)) \geq \min(\alpha(\varphi_\pi))$

Table 1: Specification of arithmetic constraints from the classical planning problem data: initial and final states and the operators; i_s and f_s stand for the initial and final state, respectively, I_O and F_O for the initial and final operator, M labels the maximum number of operators and _ stands for "unspecified" and can be understood as an anonymous Prolog variable. Finally note that since Add(O)∩Del(O)= $\emptyset$ then $\forall \varphi_\alpha \varphi_\delta \ \varphi_\alpha \neq \varphi_\delta$.

and

$$\max(\alpha(\varphi_\delta)) > \min(\alpha(\varphi_\alpha)) - 1$$

respectively corresponding to the disjoint cases ($O_\alpha = O$) and ($O_\alpha \neq O$) in Table 1 have been gathered into the single constraint $\max(\alpha(\varphi_\delta)) \geq \min(\alpha(\varphi_\alpha)) - 1$ because the choice of making an operator the activator of a formula can only be made when inserting the operator in the plan.

A Planning Procedure

The idea behind the planning procedure is the following: choose the relevant operators for the planning problem, and then solve the constraints they generate.

Of course, the choosing process must not correspond to building the solution with POP techniques but really mean "choosing relevant operators". This choice is related to activation links: to any formula involved in the solving of the problem, there is an activation link. The activation link is *incomplete* when either the activator or the deactivator of the formula is unknown; it is *complete* when both are known. The choice must make some activation links (possibly all) complete.

POP is a precondition-based process whereas the planning process just outlined only deals with both the add list and delete list. How can we then ensure that "relevant operators" also are the good ones (that is, if we don't concentrate on preconditions, operators won't be applicable to situations)? An operator also generates constraints related to its preconditions, thus taking preconditions into account.

The planning process begins with the final and initial state formulas. There exists an activation link for all these formulas; some belong to both states, some not. The formulas which belong only to either the initial state or else the final state correspond to an incomplete activation link: either the establisher or else the clobberer is missing.

An operator, from the planning problem data, is chosen to make an activation link complete. Then the chosen operator generate constraints and introduces new formulas and new (possibly incomplete) activation links. Variables of the aCSP (i.e., the min and max of the activation links) are also generated and if we want to find a solution, all these variables must be constrained: they must be involved in some arithmetic binary constraints. As seen for PutOn(X,Y,Z) and Newtower(X,Z), an operator generates no constraint for min-variables related to formulas of the delete list.

So when does the planning process end? It does end when all the variables are involved in an arithmetic binary constraint *and* all the activation links are complete (that's a theorem).

At this point, an aCSP has been built. We must solve it. A bound consistency solver is called. If the constraints are satisfiable, we have a solution. Otherwise, the procedure backtracks to the latest choice.

Here is the planning procedure set up; V is the set of variables (i.e., min and max of activation links: $\min(\alpha(\varphi))$ and $\max(\alpha(\varphi))$), C is a conjunction of arithmetic binary constraints on the variables, M is the maximum value of the final state (i.e the maximum value for any $\max(\alpha(\varphi))$) and A is the set of current activation links:

$\text{A} \leftarrow \emptyset$; $\text{V} \leftarrow \emptyset$; $\text{C} \leftarrow \emptyset$; $M \leftarrow 0$
for all $\varphi \in (i_s \cup i_f)$ **do**
  if $\varphi \in i_s$ **then**
    $\text{V} \leftarrow \text{V} \cup \{(\min(\alpha(\varphi)), I_O, D_{\min(\alpha(\varphi))})\}$
    $\text{C} \leftarrow \text{C} \wedge (\min(D_{\min(\alpha(\varphi))}) = \max(D_{\min(\alpha(\varphi))}))$

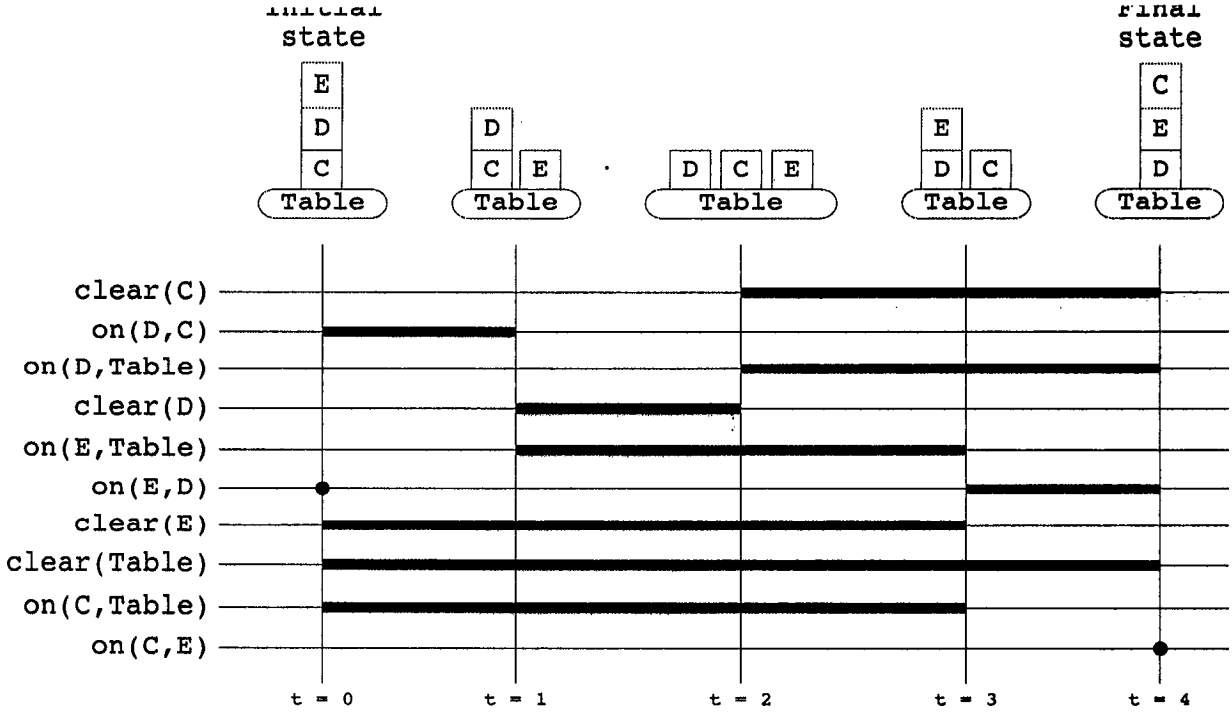

Figure 3: All the formulas involved in the transformation of the initial tower into the final tower. Successive situations are labelled with successive integers starting with zero for the initial situation. Thick lines and thick circles indicates when the aligned formula is true, i.e., is activated. There are two "special" formulas: `clear(Table)` and `on(E,D)`. `clear(Table)` is always true whatever the blocks world problem: there is a thick line indicating it is activated from the initial state to the final state. `on(E,D)` is activated during two disjoint ranges: (i) the initial state and (ii) from the fourth state to the final state.

$\wedge(\min(D_{\min(\alpha(\varphi))}) = 0)$
if $\varphi \in i_f$ **then**
A $\leftarrow$ A$\cup\{\alpha(\varphi, I_O, F_O)\}$
V $\leftarrow$ V$\cup\{(\max(\alpha(\varphi)), F_O, D_{\max(\alpha(\varphi))})\}$
C $\leftarrow$ C$\wedge(\min(D_{\max(\alpha(\varphi))}) = \max(D_{\max(\alpha(\varphi))}))$
$\wedge(\max(D_{\max(\alpha(\varphi))}) = M)$
else
A $\leftarrow$ A$\cup\{\alpha(\varphi, I_O, _)\}$
V $\leftarrow$ V$\cup\{(\max(\alpha(\varphi)), _, D_{\max(\alpha(\varphi))})\}$
C $\leftarrow$ C$\wedge(\min(D_{\max(\alpha(\varphi))}) \leq \max(D_{\max(\alpha(\varphi))}))$
end if
else
A $\leftarrow$ A$\cup\{\alpha(\varphi, _, F_O)\}$
V $\leftarrow$ V$\cup\{(\min(\alpha(\varphi)), _, D_{\min(\alpha(\varphi))})\}$
C $\leftarrow$ C$\wedge(\min(D_{\max(\alpha(\varphi))}) \leq \max(D_{\max(\alpha(\varphi))}))$
$\wedge(\min(D_{\min(\alpha(\varphi))}) > 0)$
V $\leftarrow$ V$\cup\{(\max(\alpha(\varphi)), F_O, D_{\max(\alpha(\varphi))})\}$
C $\leftarrow$ C$\wedge(\min(D_{\max(\alpha(\varphi))}) = \max(D_{\max(\alpha(\varphi))}))$
$\wedge(\max(D_{\max(\alpha(\varphi))}) = M)$
end if
end for all
return success **when** BoundsConsistency(V,C)

The set up ends by checking whether the constraint network is bounds consistent; if so, it's because the final state is included in the initial state: all activation links are complete and there is no need for planning. If not, then some activation links are incomplete and the planning process begins, adding operators, generating their variables and constraints according to Table 1 and then eventually checking for Bounds Consistency when all variables are constrained *and* all the activation links are complete. A, V, C and M come from the set up; FC is the set of yet unconstrained formulas:

FC $\leftarrow \{\varphi | \min(\alpha(\varphi)) \notin C \vee \max(\alpha(\varphi)) \notin C\}$
While FC$\neq \emptyset$ **and** A is incomplete **Do**
Choose Op s.t. $\varphi \in$FC$\wedge\varphi \in$Op
A $\leftarrow$ A$\cup$Activation links from Op
V $\leftarrow$ V$\cup$Variables from Op
C $\leftarrow$ C$\wedge$Constraints from Op
$M \leftarrow M + 1$
UpDate FC with the φs of Op
end While
If BoundsConsistency(V,C) **then**
return Success
else Backtrack to the latest choice
end if

Since the φs belong to operators it's immediate to build a totally ordered plan from the variables and their domains.

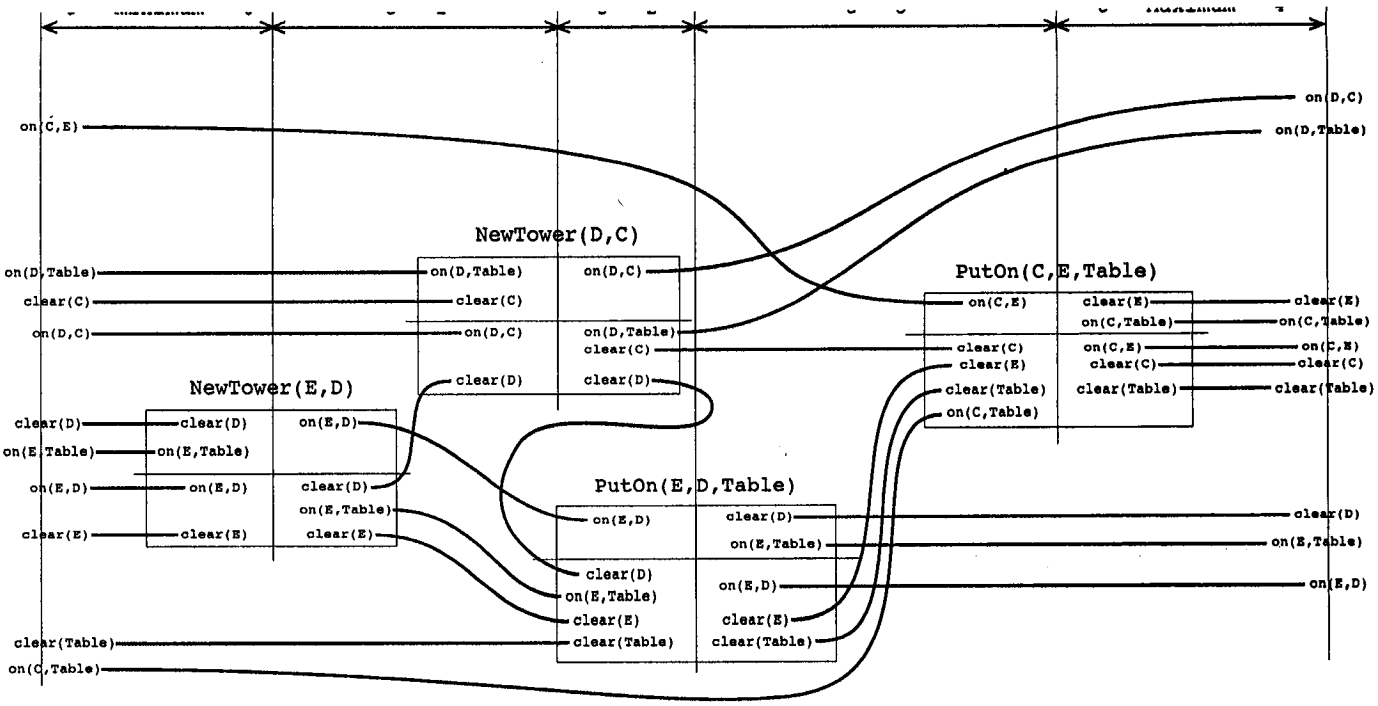

Figure 4: A solution plan, 1-cell in the bicategory of plans; the constraints in Table 1 are derived from the coherence axioms of the bicategory. An operator is a *rectangle* whose (i) left parts contain *before* formulas, (ii) right parts contain *after* formulas, (iii) the upper parts contain the deactivated formulas and (iv) the lower parts the activated formulas. There is a bijection from the before formulas to the after formulas. Links between formulas state the propagation of activation or else deactivation; internal links stating the bijection between the before and after formulas of an operator are omitted. The sets of leftmost and rightmost formulas are supersets of the initial and final state, respectively, of the classical planning problem and can be cast into a rectangle operator.

Discussion

The planning procedure above has been implemented in Open-Prolog (BRADY 1988 1998) and has been tested against a few blocks world problems within the Pweak system (JACOPIN 1994 1999).

The idea here was not to present extensive testing of an efficiently implemented (classical) planning procedure but rather to show how one could cope with the automatic transformation of a classical planning problem and the CSP framework, in merging the structures the former (operators and plans) into the components (variables, domains and constraints) of the latter.

A first important point is that the plan of Figure 4 is an 1-cell in a bicategory; this result is a theorem and not a definition: bicategorical coherence axioms (BÉNABOU 1967) must be checked. Details are outside the scope of this paper and the reader should refer to (PENON To appear). A second important point is that this result was established before the specification of the above constraints: the bicategorical results entailed the constraints, and then we tried to take advantage of these and designed the above planning procedure. A final point is that we ended up with a bicatogory when looking for plan morphisms (i.e., plan transformation) and plan composition, starting with plans like the one of Figure 2, imposing mathematical properties such as associativity and identity.

Several things are to be done, beyond the simple implementation problem (efficiency, use of commercial packages, comparisons with other planning techniques and planners, etc): we need to study the relations between the monoidal categories involved in linear logic and our bicategorical framework; this could be an interesting path to the implementation of a theorem prover in the multiplicative fragment of linear logic, which is relevant to classical planning (JACOPIN 1993; FRONHÖFER 1997); we need to lift up our procedure from propositional to atomic formulas; we also need a better close-up at the categorical structure of CSPs, e.g., (SARASWAT 1992), and its relation to planning.

In the classical version of the blocks world (CHAPMAN 1987), `clear(Table)` has a domain-dependent property: whatever the problem, it's always true. We haven't looked at domain dependent constraints; a quick way is to post them during set up, but it's something we need to investigate further with relation to the bicategorical structure. The current procedure does not take advantage of domain-dependent constraints.

The automatic transformation of a classical planning problem into an aCSP has not recently drawn much

direction whereas manual transformation recently has provide successful results in the development of the CPlan planner (VAN BEEK & CHEN 1999). However CPlan uses a state-based model and not a POP-based model as is here presented. Variables in CPlan denote states and the variables of the planning operators; whereas our variables denote the activation links, related to the formulas of the operators; that is, our model does not change the structure of the POP problem. Probably the closest work is that of (ALLEN & KOOMEN 1983); however, we consider operators as a whole and do not decompose them into formulas; our constraints correspond to the *Starts*, *Finishes* and *Overlaps* symbolic constraints on intervals, but we abstract them into their bounds in an arithmetic CSP and not use them as such in a symbolic network. Finally our framework is category theory and not temporal logic.

Note eventually that minimizing M can be cast into an objective function and then the whole framework falls in the integer linear programming basket (VOSSEN *et al.* 1999). The minimization of M is the way to produce partially ordered plan, but that's another story.

References

ALLEN, J., and KOOMEN, J. 1983. Planning using a temporal world model. In *Proceedings of IJCAI'83*, 741–747.

BÉNABOU, J. 1967. Introduction to bicategories. In *Report of the Midwest Category Seminar*, volume **67** of *Lectures Notes in Mathematics*. Springer-Verlag. 1–77.

BRADY, M. 1988–1998. Open-prolog. Web page. `http://www.tcd.ie/open-prolog/`

CHAPMAN, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* **32**(3):333–377.

FRONHÖFER, B. 1997. Plan generation with the linear connection method. *INFORMATICA, Lithuanian Academy of Sciences* **8**(1):3–22.

JACOPIN, E. 1993. Classical ai planning as theorem proving: The case of a fragment of linear logic. In *AAAI Fall'93 Symposium on Automated Deduction in Nonstandard Logics, Technical Report FS-93-01*. AAAI Press.

JACOPIN, E. 1994–1999. Pweak. Web page. `http://www-poleia.lip6.fr/~jacopin`

KAMBHAMPATI, S.; KNOBLOCK, C.; and YANG, Q. 1995. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* **76**(1-2):167–238.

KAMBHAMPATI, S. 1992. Characterizing multi-contributor causal structures for planning. In *Proceedings of AIPS'92*, 116–125.

MARRIOTT, K., and STUCKEY, P. 1998. Programming with constraints: An introduction. MIT Press.

MCALLESTER, D., and ROSENBLITT, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI'91*, 634–639.

PENON, J. To appear. A bicategorical model for articial intelligence classical planning. *Les cahiers de topologie* (In french).

SARASWAT, V. 1992. The category of constraints systems is cartesian-closed. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, 341–345.

VAN BEEK, P., and CHEN, X. 1999. Cplan: A constraint programming approch to planning. In *Proceedings of the 16th National Conference on AI.*

VOSSEN, T.; BALL Michael; LOTEM, A.; and NAU, D. 1999. On the use of integer programming models in ai planning. In *Proceedings of IJCAI'99*, 304–309.