A Framework for Hybrid and Analogical Planning

Max Garagnani
Department of Computing
The Open University
Walton Hall, Milton Keynes
MK7 6AA
United Kingdom

KEYWORDS: Planning; Domain Description Languages; Modelling Languages; Knowledge Models; Knowledge Representation and Reasoning; Diagrammatic, Analogical and Hybrid Representations.

A Framework for Hybrid and Analogical Planning

This chapter describes a model and an underlying theoretical framework for *hybrid* planning. Modern planning domain-description formalisms are based on purely *sentential* languages. Sentential representations produce problem encodings that often require the system to carry out an unnecessary amount of trivial deductions, preventing it from concentrating all the computational effort on the actual search for a plan and leading to a loss in performances. This chapter illustrates how techniques from the area of knowledge representation and reasoning can be adopted to develop more efficient domain-description languages. In particular, experimental evidence suggests that the adoption of *analogical* descriptions can lead to significant improvements in planning performance. Although often more efficient, however, analogical representations are generally less expressive than sentential ones. The hybrid approach proposed here provides a framework in which sentential and analogical descriptions can be integrated and used interchangeably, thereby overcoming the limitations and exploiting the advantages of both paradigms.

INTRODUCTION

'Planning' is the process of deciding which course of action to undertake in order to achieve a future state of affairs (*goal*) which does not hold in the present situation. Planning our daily activities, a trip, a political campaign or a military operation are just a few of the countless examples. We take a *planning domain* to be an abstract, simplified description of the world, consisting of a set of possible world *states* and a set of possible *actions* for transforming a state into another one. A planning *problem* (or planning *instance*) is specified by providing a planning domain, an *initial* state and a goal. Solving a planning problem requires finding a sequence of actions (*plan*) that will (or is expected to) transform the initial state into one in which the given goal is achieved.

Clearly, an automatic system for the solution of planning problems must be to able to internally *represent* states, actions and goals. In particular, in order to build an automated planning system, one must provide at least the following elements: (1) a *syntax* for representing world states, goals and actions; (2) a general algorithm Θ for calculating the state description $s'=\alpha(s)$ resulting from applying any action description α to any given state description *s*; and (3) a general algorithm Γ for deciding whether any goal description *G* holds (or is *satisfied*) in a given state description. Given these elements, an automatic system can use algorithm Θ to *interpret* any of the action descriptions and apply them so as to transform the initial state representation into other state representations, while algorithm Γ can be used to determine whether the assigned goal has been achieved.

In view of the above considerations, the *representation* adopted by an automated planner for modelling world states, goals and actions appears to be of crucial importance in determining the effectiveness and efficiency of the planning process. Although in the last decade the field of knowledge representation and reasoning has witnessed the birth of several new formalisms (among others, qualitative reasoning (Forbus, 1995; Forbus, Nielsen, & Faltings, 1987, 1991), semantic networks (Lehmann, 1992; Sowa, 1984), and diagrammatic representations (Glasgow, Narayanan & Chandrasekaran, 1995; Kulpa, 1994)), planning research has generally failed to assimilate and exploit such developments. In particular, the modelling languages for reasoning about action have remained, since their origins, purely *sentential* (i.e., textual, or based on predicate and propositional logic) (McCarthy & Hayes, 1969; Fikes & Nilsson, 1971; Pednault, 1989; McDermott, Knoblock, Veloso, Weld & Wilkins, 1998). Even the most recent version of PDDL, the *de facto* standard planning domain description language (Fox & Long, 2003) requires the domain modeller to describe all aspects of a problem (including spatial and topological relations) using only sets of propositions.

The rest of this introductory section argues that, although very expressive and flexible, sentential languages are often *inefficient*[1] for describing and solving even simple planning problems. In particular, sentential planning representations tend to produce inefficient encodings of domains that involve the *movement* of a number of distinct objects subject to even simple spatial constraints. The remainder of the chapter is divided into two main parts: the first one, containing two sections, introduces *analogical* planning representations and illustrates, with a simple example and then through the analysis of an actual implementation, how such formalisms can help overcome some of the shortcomings of sentential descriptions. The second part, entirely contained in one section, proposes a framework for hybrid planning, in which sentential and analogical descriptions are integrated. These two parts are joined by a transitional section ("Characterising Analogical Models"), which provides some background on analogical formalisms and analyses their differences from sentential ones. The two final sections discuss related work, advantages and limitations of the approach.

Planning with Sentential Representations: a Simple Example

In order to introduce sentential representations, let us begin with a simple example, based on a variation of the classical Blocks-World (BW) planning domain. The BW domain consists of a robot arm able to pick up and put down blocks that lie on a table. In the classical version of the domain, the blocks are all identical. Here, blocks can have different weights, and a block can only be picked up from the top of a stack if the stack contains at least another block which is heavier than the one being removed. The possible actions of this domain are *stack* and *unstack*: *Stack*(*x*,*y*) consists of picking up a block *x* (of any weight) from the table and stacking it onto another block *y*; *Unstack*(*x*,*y*) picks up a block *x* currently lying on another block *y* and puts it on the table (subject to the stack containing a block heavier than *x*). Figures 1.(*a*) and 1.(*b*) depict, respectively, the initial state and goal for an example of BW planning problem (the weights of the blocks are left unspecified).

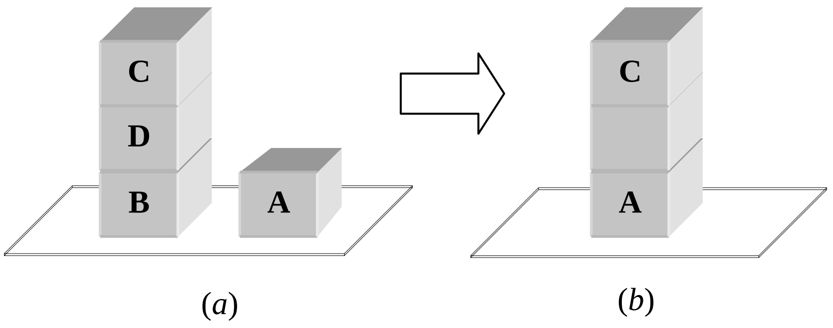


Figure 1. Simple Blocks-world problem: (*a*) initial state; (*b*) goal.

The goal describes the arrangement of blocks A and C, but does not specify the final position of B or D, although it does require that one of them be positioned between A and C. Also notice that, unlike the original BW, this version allows the formulation of problems for which no plan solution exists.

Most modern planning systems would describe this domain using a sentential formalism not too different from the original STRIPS (Fikes & Nilsson, 1971). In such model, the current world s*tate* is represented as a set of *ground atoms* (atomic logical formulæ) of the form $p(x_1,\ldots x_k)$, where p is a predicate with k arguments. An atom A is said to *hold* in a state s iff $A\in s$. A negative atom $\neg A$ holds in s iff A does not hold in s. A *literal* is an expression of the form A or $\neg A$, where A is an atom. For example, consider a language with predicates `On(`x,y`)`, `Heavier(`x,y`)` and `Clear(`x`)`, where x, y vary on the *constant* symbols {`A`, `B`, `C`, `D`, `Table`}, representing the corresponding objects of the domain. The initial state of Figure 1.(*a*) can be described by the following set of ground atoms:

$$s_0 = \{\mathtt{On(A,Table)}, \mathtt{Clear(A)}, \mathtt{On(B,Table)}, \mathtt{On(D,B)}, \mathtt{On(C,D)}, \mathtt{Clear(C)}, \\ \mathtt{Heavier(A,B)}, \mathtt{Heavier(B,C)}, \mathtt{Heavier(C,D)}\}$$

In this example, blocks `A`,`B`,`C` and `D` are in order of decreasing weight. Notice that the two ground atoms `Heavier(A,C)`, `Heavier(B,D)` have not been included, although they should hold in s_0 in virtue of the transitivity property. This property can be imposed on the relation `Heavier` through the addition of the following *domain axiom* to the description (in which all the free variables are implicitly universally quantified):

$$(\rho_1) \qquad \mathtt{Heavier}(x,y) \land \mathtt{Heavier}(y,z) \rightarrow \mathtt{Heavier}(x,z)$$

Axiom (ρ_1) avoids having to explicitly include in s_0 all the instances of `Heavier` that hold in the initial state, which would be unwieldy for large numbers of blocks. However, as discussed below, the introduction of domain axioms in the domain description should not be taken too lightly, as it can have a negative impact on planning performance. Incidentally, domain axioms are also useful for describing the actions and the goal G_1 (Figure 1.(*b*)):

$$G_1 = \{\mathtt{On(A,Table)}, \mathtt{Above(C,A,2)}, \mathtt{Clear(C)}\}$$

The "derived" predicate `Above(`x,y,n`)` denotes that block x is the n-th block above y, and is defined in terms of the "basic" predicate `On(`x,y`)` through the following domain axioms (where $n\in\aleph$, the set of natural numbers, and variables x,y and z represent distinct blocks):

$$(\alpha_1) \qquad \mathtt{On}(x,y) \rightarrow \mathtt{Above}(x,y,1)$$
$$(\alpha_2) \qquad \mathtt{On}(x,y) \land \mathtt{Above}(y,z,n) \rightarrow \mathtt{Above}(x,z,n+1)$$

In view of the presence of domain axioms, the previous definition of 'hold' needs to be extended: an atom A *holds* in a state s iff either $A\in s$, or A can be derived from domain axioms whose premises (left-hand side) hold in s (for a more precise and formal definition of the semantics of domain axioms in planning, see (Thiébeaux, Hoffmann & Nebel, 2003)).

The possible actions of the domain are described by parameterised *operators* $P\Rightarrow E$, consisting of *preconditions* P and *effects* E. These are sets of parameterised literals. For example, *Stack*(x,y) and *Unstack*(x,y) would typically be encoded as follows:

Stack(x,y)	-- *Picks up block x from the table and puts it onto block y* ($x\neq y$)
Parameters:	x,y – `Block`
Preconditions:	{`On(`x`, Table)`, `Clear(`x`)`, `Clear(`y`)`}
Effects:	{`On(`x,y`)`, $\neg$`On(`x`, Table)`, $\neg$`Clear(`y`)`}

Unstack(*x*,*y*)	-- *Picks up x from y and puts it on the table*
Parameters:	*x*,*y* – `Block`
Preconditions:	{`On`(*x*,*y*), `Clear`(*x*), `Above`(*x*, *z*, *n*), `Heavier`(*z*, *x*) }
Effects:	{¬`On`(*x*,*y*), `Clear`(*y*), `On`(*x*, `Table`)}

All the variables in the preconditions are implicitly existentially quantified[2]; the type `Block` consists of the set of symbols {`A`,`B`,`C`,`D`}. The semantics of these operators (Lifschitz, 1990) is as follows: for an operator to be applicable to a state *s*, all its preconditions must *hold* in *s*. When an operator is applied to a state *s*, all the positive atoms of the effects are *added* to *s*, and all the negative atoms of the effects are *deleted* from *s*. For example, the action *Stack*(`A`,`C`) is applicable in state s_0, and its application would produce the following state s_1:

$$s_1 = \{\texttt{On(B, Table)}, \texttt{On(D, B)}, \texttt{On(C, D)}, \texttt{On(A, C)}, \texttt{Clear(A)}, \texttt{Heavier(A, B)}, \texttt{Heavier(B, C)}, \texttt{Heavier(C, D)}\}$$

Notice that the set of instances of `Heavier` that hold in the initial state remains constant throughout the solution of the problem. This is a consequence of the fact that no instance of the predicate is ever affected – directly or indirectly – by the operators; this property of the domain can be automatically detected and exploited by modern planners to restrict the search to the parts of the problem that can actually change, avoiding unnecessary calculations.

Unfortunately, unlike `Heavier`, the instances of `Above`, although not appearing in any of the operator effects, do change as an *indirect* consequence of changes in the `On` relation. In particular, *any* movement of the blocks causes the set of ground instances of `Above` currently holding in the state to change. For example, consider the instances of `Above` that hold in state s_0 (Figure 1.(*a*)), derived using axioms (α_1),(α_2) as follows:

(1.1)	`On(D,B)` $\rightarrow$ `Above(D, B, 1)`	(from (α_1))
(1.2)	`On(C,D)` $\rightarrow$ `Above(C, D, 1)`	(from (α_1))
(1.3)	`On(C,D)` $\wedge$ `Above(D, B, 1)` $\rightarrow$ `Above(C, B, 2)`	(from (α_2) + (1.1))

After the application of *Stack*(`A`,`C`), the instances of `Above` holding in state s_1 are:

(1.4)	`On(A,C)` $\rightarrow$ `Above(A, C, 1)`	(α_1)
(1.5)	`On(C,D)` $\rightarrow$ `Above(C, D, 1)`	(α_1)
(1.6)	`On(D,B)` $\rightarrow$ `Above(D, B, 1)`	(α_1)
(1.7)	`On(A,C)` $\wedge$ `Above(C, D,1)` $\rightarrow$ `Above(A, D, 2)`	(α_2)+ (1.5)
(1.8)	`On(C,D)` $\wedge$ `Above(D, B,1)` $\rightarrow$ `Above(C, B, 2)`	(α_2)+ (1.6)
(1.9)	`On(A,C)` $\wedge$ `Above(C, B, 2)` $\rightarrow$ `Above(C, A, 3)`	(α_2)+ (1.8)

This is a first indication that the introduction of domain axioms in the problem may lead to significant amounts of additional computation. This is argued in more details below.

The Inefficiencies of Planning with Domain Axioms

Let us consider how a forward state-space planner would solve a problem in the BW domain described earlier. The truth of the derived predicate `Above`(*x*,*y*,*n*) can be deduced from the current state at any point of the planning process using (α_1),(α_2). However, whenever any instance of the `On` predicate changes, it is necessary for the planner to re-calculate the `Above` relation, as the truth of some of its instances will have been affected by the change. The number of steps necessary to derive all the instances of the relation for a tower of *m* blocks is equal to $(m(m-1))/2$. Hence, in a BW domain with *m* blocks, the calculation potentially requires $O(m^2)$ extra steps after each move. In general, if the relation to be deduced contains *k* – instead of only 2 – arguments varying on a set of *m* objects (constant symbols) of the domain, the number of steps required is $O(m^k)$. While in some simple cases (like this one) the number of deductions can be reduced by recalculating only those instances strictly affected by the action (Pednault, 1989; Davidson & Garagnani, 2002), a forward-search algorithm able to deal with *any* arbitrary set of domain axioms may have to carry out, in the worst case, a number of steps *exponential* in the size of the domain description (if the arity of the axioms is a measure of this size), or *polynomial* in the number of objects (if the arity is a constant), after each operator application and for each relation affected (Thiébeaux *et al*., 2003).

A *backward*-search algorithm would incur in similar (or even worse) problems. For clarity and generality of the analysis, let us rename predicates `Above(`*x,y,n*`)` and `On(`*x,y*`)` as $D(x,y,n)$ and $B(x,y)$, for "derived" and "basic", respectively. The two axioms then become:

$$(\beta_1) \qquad B(x, y) \rightarrow D(x, y, 1) \qquad (x \neq y)$$
$$(\beta_2) \qquad B(x, y) \wedge D(y, z, n) \rightarrow D(x, z, n+1) \qquad (x \neq y \neq z \neq x)$$

All the occurrences of the two predicates in the operators, initial and goal-state are similarly renamed. Now, consider, for example, the problem of establishing (achieving) the preconditions of the *Unstack* operator. Suppose that the term $D(x,y,n)$ is picked first, and that its variable *n* is still unbound. Since all the operators contain, in their effects, only basic predicates, the only way to discover how this term can be achieved consists of transforming it into an equivalent expression containing only $B(x,y)$ terms. If *n* is unbound, a direct transformation is not possible, as *n* could be any positive integer. However, a sufficiently sophisticated planning system might be able to recognise that, if the problem contains only a *finite* number of blocks, the range of *n* is finite. For example, in presence of only four blocks, the planner should be able to apply domain axioms (β_1), (β_2) to transform the expression $D(x,y,n)$ into the following equivalent disjunction of conjunctive terms:

$$(2.0) \qquad B(x,y) \vee (B(x,w) \wedge B(w,y)) \vee (B(x,v) \wedge B(v,w) \wedge B(w,y))$$

Unfortunately, even assuming that this is possible, the introduction of disjunctive expressions like (2.0) would lead to a significant increase in the branching factor of a backward search, having negative effects on the performance. Notice that while this simple example causes the branching factor to grow "only" by a factor $m-1$ (where *m* is the total number of blocks), domains containing more and/or more complex axioms (e.g., involving multiple linear or non-linear recursions (Han, 1989)) would require rather complex transformations of the derived predicates and lead to much higher branching factor increases. Finally, notice that even simple domains like BW can involve several complex axioms. For example, Cook and Liu (2003) provide an axiomatization of BW using seven different recursive axioms only to describe the 'on' relation (which they call *Above*), and demonstrate that every decision procedure for the resulting theory must take at least exponential time.

Domain Axioms and the Ramification Problem

The problem of domain axioms in planning appears to be closely related to the so-called *ramification* problem (Georgeff, 1987) in automated reasoning. Pollock (1998) accurately describes this problem as one that arises from the observation that

"[..] *in realistically complex environments, we cannot formulate axioms that completely specify the effects of actions or events.* […] *in the real world, all actions have infinitely many ramifications stretching into the indefinite future. This is a problem for reasoning about change deductively* […]" (p.536)

Using one of Pollock's examples, among the effects of striking a match we should include such things as "displacing air around the match, marginally depleting the ozone layer, raising the temperature of the earth's atmosphere, marginally illuminating Alpha Centauri, […], etc." (p. 537). Naturally, a planning domain description is not expected to model *all* such complex ramification of events and actions: planning involves reasoning about a simplified version of the real world. However, even very simple, toy-like domains such as BW can already involve several complex domain axioms (Cook & Liu, 2003). If the target domain considered is a real application, the model is likely to contain tens of axioms and very large numbers of objects (e.g., see the "PSR" domain in (Bonet & Thiébeaux, 2003)).

In order to address the problem of planning in presence of domain axioms, some researchers (e.g., (Gazen & Knoblock, 1997; Davidson & Garagnani, 2002; Thiébeaux *et al*., 2003)) have developed *pre-processing* techniques for automatically transforming a planning problem into an equivalent one that does not contain axioms, and which can be solved using simple and efficient planning algorithms. Unfortunately, recent theoretical results demonstrate that any attempt to compile away an arbitrary set of domain axioms leads, in general, to equivalent planning problems having *exponentially* longer plans (in the number of objects and arity of the axioms) (Thiébeaux *et al*., 2003). According to such results, if the maximum arity of all the predicates of the language is a constant, the growth in plan length is only polynomial. However, from a practical point of view, even a polynomial blow up of the plan-solution length forces a planning algorithm to carry out a significantly larger amount of search to discover such plan. Indeed, even for simple BW problems, experimental evidence (Davidson & Garagnani, 2002) shows that the planning performance on pre-processed problems depends much on the specific domain axioms, pre-processing technique and algorithm adopted, and is often worse than that obtained with planners that are able to solve the original problem directly (Thiébeaux *et al*., 2003).

An alternative planning paradigm which would appear particularly suited for dealing with domain axioms is that of planning as *satisfiability*, or "SAT-based" planning (Ernst, Millstein and Weld, 1997; Kautz & Selman, 1992; 1999). A SAT-based planning system transforms a planning problem description into a *propositional logic formula* which, if satisfied, implies the existence of a plan solution.[3] The use of additional domain axioms in such a framework is quite natural, as axioms are treated simply as propositional logic formulæ. However, as Wilkins and des Jardins (2001) observed, "additional knowledge encoded as axioms may increase the size of the problem and make the problem even harder to solve" (Wilkins and des Jardins, 2001, p.109). This is confirmed by experimental evidence (Davidson & Garangnani, 2002), indicating that whether the addition of domain axioms helps or hurts may depend on the particular combination of axioms, problem and planning algorithm (Kautz & Selman, 1998).

In summary, the presence of domain axioms, closely related to the ramification problem, appears to be inevitable in *sentential* descriptions of realistically-complex domains, and to represent the potential cause for severe decreases in planning performance. The next section illustrates how analogical models can, in many cases, completely eliminate this problem, by making domain axioms *implicit* in the representation of the world.

INTRODUCING ANALOGICAL PLANNING

Planning in realistic domains is closely related to the problem of common-sense reasoning (McCarthy, 1958). In this context, several researchers have argued for the need of formalisms that allow a more direct (or "vivid") representation than traditional sentential descriptions (e.g., (Halpern & Vardi, 1991; Levesque, 1986; Khardon, 1996)). In particular, analogical and diagrammatic representations have long been of interest to the knowledge representation community (Amarel, 1968; Sloman, 1975; Hayes, 1985) (see (Kulpa, 1994) for a review, and (Glasgow *et al.*, 1995) for a representative collection). In order to clarify the main ideas behind such descriptions, we begin with an example of diagrammatic[4] planning domain description. A more general discussion on the properties of analogical models and on how they differ from sentential ones is given later on, in the fourth section of this chapter.

SetGraphs in a Nutshell

Consider a representation in which a state is a directed *graph* where the vertices are (possibly labelled) *sets* of symbols. This type of representation will be called *setGraph*. Figure 2.(*a*) is an example of a setGraph encoding a BW state with three blocks and one table, represented by symbols 'A', 'B','C', 'Table'. The vertices of the graph are depicted as ovals. The edges of the graph (bold arcs) represent 'on' relations between spatial locations: if a vertex containing *x* is linked to a vertex containing *y*, then `On` (*x*,*y*) holds in the current state.

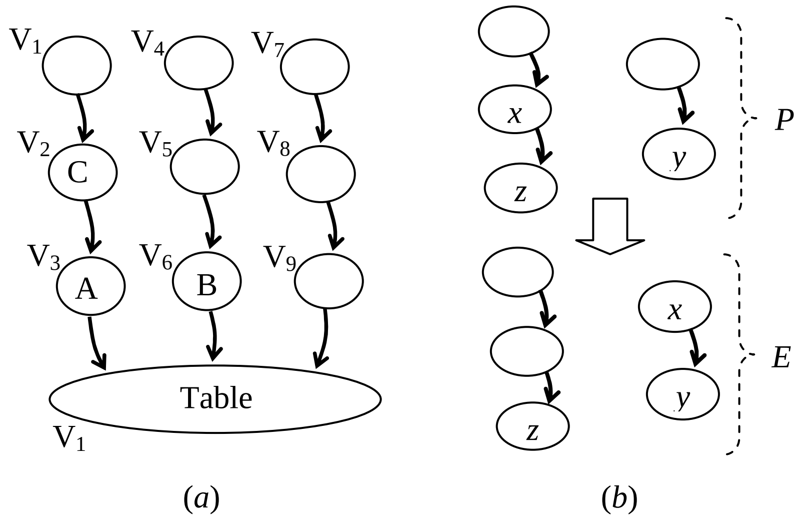


Figure 2. An analogicald model of BW: (*a*) state representation; (*b*) *Move*(*x*,*y*,*z*) operator (where $x \in \{A, B, C\}$ and $y, z \in \{A, B, C, Table\}$)

Assume that all the symbols of the graph can be moved from one vertex (set) to any other through the application of (analogical) operators, which specify the set of legal transformations of a setGraph. Figure 2.(*b*) depicts the graphical representation of an operator, $P \Rightarrow E$. The operator preconditions *P* describe a specific arrangement of symbols in a part (sub-graph) of the current state; the effects *E* describe the arrangement of these symbols in the same sub-graph after the application of the operator. Intuitively, an operator $P \Rightarrow E$ is applicable to a state (setGraph) *s* iff each of the graphs contained in *P* can "overlap" with (be mapped to) a sub-graph of *s* having the same "structure", so that each variable corresponds to a *distinct* symbol, each vertex to a vertex, each edge to an edge, and: (1) if a variable is contained in a set (vertex), the corresponding symbol is contained in the corresponding set;

(2) if an edge links two vertices, the corresponding edge links the corresponding sets; and (3) if a set is empty, the corresponding image is empty. Only if all of these conditions hold, we will say that the precondition setGraphs are *satisfied* in *s*.

Variables can be of specific *types*, subsets of the universe of symbols. For example, variable *x* of the *Move*(*x*,*y*,*z*) operator has type *Block*={A, B, C}, while *y, z*∈ *Object*={A, B, C, Table}. Thus, this operator encodes the movement of a block *x* from its current location to a new one, originally empty, situated "on top" of a set containing another block (or the table) *y*. Notice that the operator is applicable only if block *x* has an empty set on top of it (i.e., if *x* is clear).

The application of an operator to a state *s* causes the corresponding symbols in *s* to be re-arranged according to the situation described in the effects *E*. The *Move*(*x*,*y*,*z*) operator can be applied to the state of Figure 2.(*a*) in several different ways. For example, one possible binding of the variables is {*x*/C, *y*/Table, *z*/A}. The application of *Move*(*x*,*y*,*z*) with this binding would unstack block C from A and put it on the table (i.e., in set V_9 of Figure 2.(*a*)).

This simple graph-based notation can encode any "classic" BW problem. Notice that the representation is not limited to just forward state-space search planning: once the semantics of action, state and goal representation are identified, the representation can be used to find a plan using *any* search algorithm, e.g. reasoning backward from the goal to the initial state using state-space search, or plan-space search techniques. The example below illustrates how a partial order, causal-link planning algorithm (McAllester & Rosenblitt, 1991; Penberthy & Weld, 1992) can solve the Sussman anomaly (Sussman, 1990) using setGraphs descriptions.

Example 2.1 – Consider the BW problem in which the initial state *I* depicted in Figure 2.(*a*) is required to be transformed into a state in which block A is on B, B is on C, and C lies on the table (this goal is represented by the setGraph *G* in the rightmost part of Figure 3).

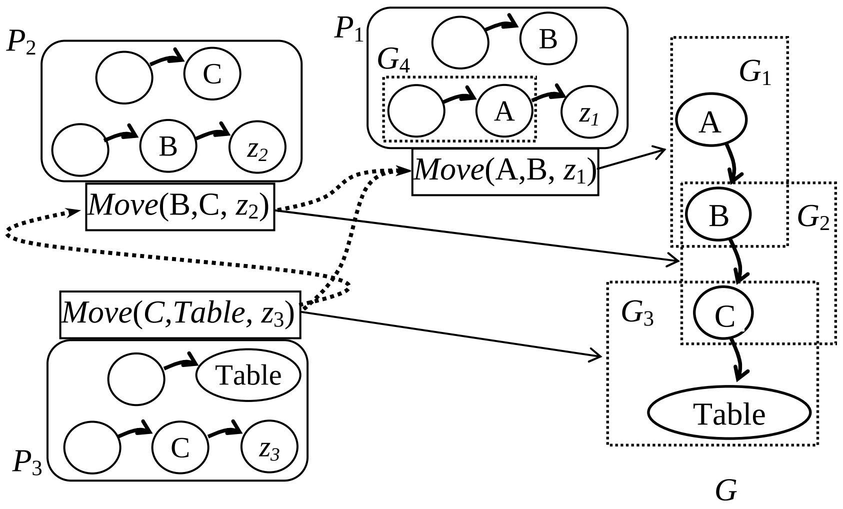
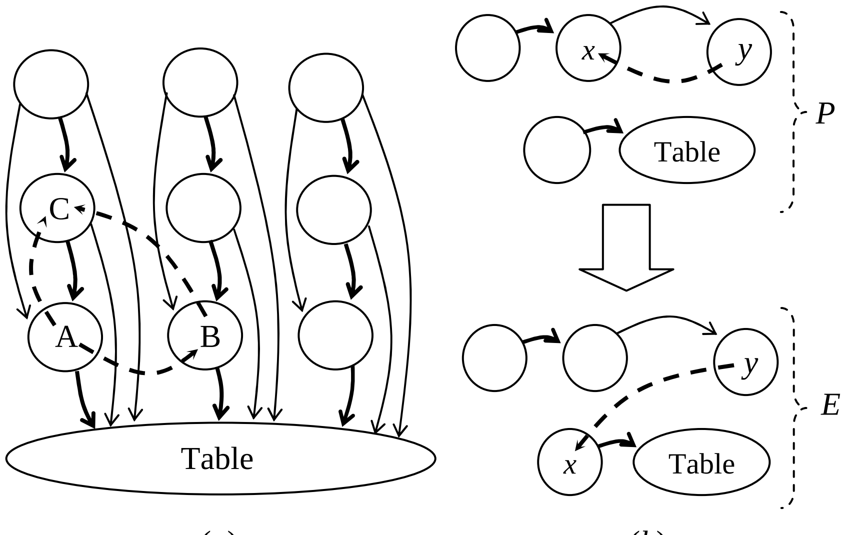


Figure 3. Solving the Sussman anomaly using causal-link diagrammatic planning (see text for details)

At the start of the planning process, goal *G* is not satisfied in the initial state *I*, and is added to in the set of unachieved goals. The algorithm then tries to find a way to achieve *G* or some of its parts (sub-graphs) by "matching" the effects of the *Move*(*x*,*y*,*z*) operator (Figure 2.(*b*)) with the goal. This reveals that the sub-graph (sub-goal) G_1 (see Figure 3) can be obtained by executing *Move*(A, B, z_1), for some object z_1. The planner then adds this step to the plan, and the preconditions P_1 – required to execute it – to the set of unachieved goals. A similar process is repeated for sub-goals G_2 and G_3, which require the addition of two other steps with preconditions P_2 and P_3, respectively (notice that the three steps added are initially unordered). At this point, the graph representing the goal *G* has been entirely "covered" by the combined effects of three *Move*(*x*,*y*,*z*) steps, and all of its elements have been "achieved". The planner can then move on to consider the unachieved setGraphs P_1, P_2 and P_3. Of these, only the sub-graph G_4, part of preconditions P_1, cannot be satisfied in the initial state *I*. The algorithm, however, discovers that G_4 can be established by the effects of step *Move*(C, Table, z_3) if object z_3 is bound to block A.[5] Hence, the planner adds the constraint z_3=A to the plan (not shown in the figure) and an ordering constraint between step *Move*(C,Table,A) and *Move*(A,B, z_1) (represented in Figure 3 by a dotted arrow). At this point, the algorithm has identified a set of steps that achieve goal *G* and whose preconditions are either satisfied in the initial state or established by another step. However, the planner must also check for other possible interactions between steps. For example, executing step *Move*(B,C,Table) before *Move*(C,Table,A) would affect the preconditions of the latter, as block C would no longer be clear. In order to prevent this type of "clobbering" effects, two further ordering relations have to be added (see Figure 3), leading to the final correct plan containing three linearly ordered steps, ⟨*Move*(C,Table,A), *Move*(B,C,Table), *Move*(A,B,Table) ⟩.

Notice that the diagrammatic representation of BW can be easily extended to encode the relations 'heavier' and 'above' between blocks (refer to the first section of this chapter). The BW state depicted in Figure 4.(*a*) extends the setGraph representation with two new types of edges, depicted as *thin* and *dashed* arcs.



(*a*) (*b*)

Figure 4. A richer diagrammatic model of BW: (*a*) current state; (*b*) *Unstack*(*x*,*y*) operator. Bold, thin and dashed arrows indicate, respectively, 'on', 'above' and 'heavier' relations (see text for details).

A thin arc linking a vertex containing symbol *x* to one containing symbol *y* indicates that `Above(`*x*,*y*,*n*`)` holds (with *n*>1); a dashed arrow linking symbol *x* to symbol *y* indicates that `Heavier(`*x*,*y*`)` holds.[6] Figure 4.(*b*) depicts the analogical version of the *Unstack*(*x*,*y*) operator (the *Stack*(*x*,*y*) operator is identical to the *Move*(*x*,*y*,*z*) operator of Figure 3.(*b*) with *z*=Table). The preconditions *P* require the existence of a block *y* such that both `Heavier(`*y*,*x*`)` and `Above(`*x*,*y*,*n*`)` hold.[7] The effects *E* encode the new position of block *x*, now located on the table. (Notice that when a symbol is moved, all arrows (edges) connected to it move with it).

An interesting property of analogical representations is that they allow the spatial relations existing between the "mobile" objects of a domain to be represented as *static* (or invariant) elements of the description, whenever the set of possible positions in which such objects can be (relatively to each other) is finite and predetermined. For example, it is easy to see that all the edges of the setGraph of Figure 4.(*a*) (including all those representing spatial relations between blocks) are static and would remain unchanged throughout any plan execution, as they are not affected by any of the possible actions. This is enabled by the distinction that the chosen analogical encoding makes between the description of the effects of action on an object's state or location and the description of the invariant relations that hold between these objects (or between the spatial locations that these objects can occupy). This encoding, however, could have been "emulated" by a sentential representation. For example, if the vertices of the setGraph of Figure 4.(*a*) were considered as entities of the domain identified by symbols $\mathtt{V}_1,\ldots,\mathtt{V}_9$ (see Figure 2.(*a*)), then this state could be described as follows:

$$\begin{aligned} I = \{ & \mathtt{On}(\mathtt{V}_1, \mathtt{V}_2), \mathtt{On}(\mathtt{V}_2, \mathtt{V}_3), \mathtt{On}(\mathtt{V}_3, \mathtt{Table}), \ldots, \mathtt{On}(\mathtt{V}_9, \mathtt{Table}), \\ & \mathtt{Above}(\mathtt{V}_1, \mathtt{V}_3), \mathtt{Above}(\mathtt{V}_2, \mathtt{Table}), \ldots, \mathtt{Above}(\mathtt{V}_7, \mathtt{Table}), \\ & \mathtt{Heavier}(\mathtt{A,B}), \mathtt{Heavier}(\mathtt{B,C}), \mathtt{Heavier}(\mathtt{A,C}), \\ & \mathtt{In}(\mathtt{C}, \mathtt{V}_2), \mathtt{In}(\mathtt{A}, \mathtt{V}_3), \mathtt{In}(\mathtt{B}, \mathtt{V}_6), \\ & \mathtt{Empty}(\mathtt{V}_1), \mathtt{Empty}(\mathtt{V}_4), \mathtt{Empty}(\mathtt{V}_5), \ldots, \mathtt{Empty}(\mathtt{V}_9) \} \end{aligned}$$

Predicate `In`(x,y) indicates that symbol x is inside vertex y; `Empty`(x) indicates that vertex x contains no symbols; `On`(x,y), `Above`(x,y) and `Heavier`(x,y) denote the corresponding edges between vertices and symbols. In the world state I, only the instances of the predicates '`In`' and '`Empty`' are subject to change; the rest of the atoms are invariant. Notice that although obtaining this encoding from Figure 4.(*a*) appears now as a straightforward task, no planner would have been able to *automatically* generate state I from the original, sentential version of BW considered earlier on, in the first section of this chapter.

It should be noticed that although states, operators and goals have been described here using a purely diagrammatic notation, the translation of such descriptions into a formal language is relatively straightforward, as all of their components (graphs and sets) have a direct mathematical representation[8] (this is illustrated in the next section). Also notice that the number of edges in a setGraph used to represent relations (like 'above' or 'heavier') between the objects of the domain grows only polynomially in the number of objects; this result could be obtained in a sentential descriptions only by keeping the arity of the predicates constant.

When compared to sentential descriptions, analogical representations appear, at first glance, more intuitive, simpler to understand and to manipulate. The use of a model that reflects the spatial and topological aspects of the real domain suggests that this type of representations should be more suitable for the application of common sense reasoning, heuristic extraction and learning techniques. For example, an operator like the one depicted in Figure 2.(*b*) should not be difficult to learn, given appropriate image-processing techniques. The next section will demonstrate how, even without exploiting the "static" properties of a domain, analogical models can be significantly more efficient than sentential ones, particularly in domains involving the movement of objects subject to spatial constraints.

ANALOGICAL PLANNING: A CASE STUDY

In order to illustrate the viability and efficiency of analogical planning, we describe an example of an implemented analogical planning domain description. The representation adopted consists of a simplified version of the setGraph model proposed in the previous section. In particular, a state consists of a finite set of *arrays* of symbols. The experimental results obtained with a prototype system that adopts such representation are briefly summarised and discussed below.

Syntax and Semantics of Array-based Planning

In order to formalise a planning domain description language, we need to specify a syntax for describing states and actions (i.e., transformations of legal states into legal states). A world state is represented here by a set of fixed-length, one- or two-dimensional arrays. The name, contents and length of a one-dimensional array are described using the following syntax:

$$A\,[a_1\,|\,a_2\,|\ldots|\,a_n\,]$$

This expression declares an array of n cells of name A and initialises its contents so that its i-th cell contains symbol a_i. The name A can be any string of characters; $a_1,\ldots,a_n$ are symbols in $U' = U \cup \{\ \_\ \}$, where U is the chosen universe of symbols and the special symbol '_' indicates an empty cell (i.e., the absence of any symbol). For example, a BW domain with three blocks could be described using three one-dimensional arrays of characters, each one representing a stack; hence, the state represented in Figure 2.(*a*) could be encoded as follows:

$$I = \{\ \texttt{s1}[\,\texttt{T}\,|\,\texttt{A}\,|\,\texttt{C}\,|\,\_\,],\ \texttt{s2}[\,\texttt{T}\,|\,\texttt{B}\,|\,\_\,|\,\_\,],\ \texttt{s3}[\,\texttt{T}\,|\,\_\,|\,\_\,|\,\_\,]\,\}$$

Bi-dimensional arrays can be similarly specified. The symbol `T` is used to represent a "table-location", and is introduced only to simplify the description of the *Move* action (see below).

In order to describe actions, let us introduce a notation for identifying and manipulating symbols within an array. The expression $A(x,y\ldots z)$, where $x,y\ldots z \in U'$ and A is a string, is said to be *satisfied* in a state s iff s contains an array with name (or of type) A such that each one of the symbols $x,y\ldots z$ appears in A at least once. The expression $A(x\,|\,y\,|\ \ldots|\,z\ )$ is satisfied in a state s iff $A(x,y\ldots z)$, and $x,y\ldots z$ are *consecutive* elements of array A. In addition, we use (possibly typed) variables to represent elements of U or array names.

The syntax adopted for analogical action descriptions is analogous to that used for sentential action description. An array-based operator $P \Rightarrow E$ consists of preconditions P and effects E, each containing a set of array expressions. An operator transforms the arrays identified in the preconditions P into the arrays described in the effects E. For example, the following represents the *Move*(x,y,z) operator for the array-based encoding of BW introduced above:

Move(x,y,z)	-- *Moves block x from object z onto object y*
Parameters:	*x – Block*; *y, z – Objects*
P:	$\{\ stack_1(\ z\,|\,x\,|\,\_\ ),\ stack_2(\ y\,|\,\_\ )\ \}$
E:	$\{\ stack_1(\ z\,|\,\_\,|\,\_\ ),\ stack_2(\ y\,|\,x\ )\ \}$

An operator is applicable to a state s only if all its preconditions are satisfied in s. Intuitively, this equates to map each array expression to an array of s and each variable to a symbol of the array such that the arrangement of the symbols "matches" that of the variables. In this example, the two variables $stack_1$, $stack_2$ can be bound to any pair of distinct array names taken from the set {`s1`,`s2`,`s3`}. Variable $x \in Block$={`A`,`B`,`C`}, while $y,z \in Objects$={`A`,`B`,`C`,`T`}. The preconditions P are satisfied if two stacks can be found which contain, respectively, a clear block x lying on an object (another block or a table-space) z, and a clear object y. The effects E describe the final arrangement of symbols x,y,z in the *same* cells of the two arrays $stack_1$, $stack_2$ identified by the preconditions P (when a symbol is moved to an empty cell, the original cell becomes automatically empty). Notice the similarity between this analogical operator and the one of Figure 2.(*b*): although the latter is described graphically and the former uses a formal (though not sentential) language, their preconditions and effects are semantically equivalent. Indeed, the above operator can solve any BW problem expressed in the array-based representation applying precisely the same steps that would be required to solve the same problem in the setGraph model (e.g., see the Sussman anomaly of Figure 3).

In order to be able to deal with two-dimensional arrays, it is possible to extend this syntax with additional symbols representing specific spatial relations between pairs of elements.[9] In spite of its simplicity, this notation reflects some of the main characteristics of the setGraph representation, and was adopted to develop a working prototype of a simple array-based planner (ABP) that was tested on a set of planning problems. The experimental results obtained (reported fully in (Garagnani & Ding, 2003)) are briefly summarised below.

Experimental Results of Array-based Planning

The notation introduced above allows the encoding of a small set of relatively simple – yet quite widely used – benchmark problems taken from the International Planning Competitions[10] and the planning literature. For the experiments, five propositional planning domains (BW, Eight-puzzle, Miconic, Briefcase and Gripper) were chosen and translated into equivalent analogical representations. In order to compare the performance of analogical planning against sentential planning, a second planner was also implemented, identical in *all* aspects to ABP except for the representation adopted. Both planners (implemented in Java) used the same, simple, breadth-first, forward state-space search algorithm, and were run on the same machine to solve exactly the same problems. However, while ABP reasoned using an array-based notation, the second, sentential, planner (SP for short) adopted a classical, propositional (STRIPS-based) language, with types. Importantly, each of the domains was translated so as to present, in its analogical version, exactly the *same search space* as in the propositional version (i.e., the two state spaces originated for any one problem of the domain have the same *cardinality* and the same *structure*). For each domain, several planning problems were solved by both ABP and SP. Figure 5 contains the actual domain description of the BW domain used by ABP in the experiments, and the encoding of the Sussman anomaly problem instance. The syntax adopted parallels the notation of the current (sentential) standard planning domain description language, PDDL (Fox & Long 2003) (for a complete BNF formalisation of the syntax adopted by ABP, see (Garagnani & Ding, 2003)).

The results (reported fully in (Garagnani & Ding, 2003)) demonstrate a clearly superior performance of ABP on *all* of the five domains, and on *all* of the problem instances. Tables 1 and 2 contain the time required by the two planners for solving the same problems in Gripper and BW, respectively. (All the problems are taken from the set of problems used in the classical track of the 2000 International Planning Competition). The speed-up factor varied from two to as much as one hundred and sixty times faster (e.g., see problem *4–1* in Table 1).

	1–0	*2–1*	*2–1*	*3–0*	*2–2*	*3–1*	*4–0*	*4–1*
SP	0.0	0.3	5.1	31	167	862	1866	(>16 *hours*)
ABP	0.0	0.0	0.4	1.1	2.8	13	26	354

Table 1. Planning time (*s.*) for Gripper problems ("*m–n*"= *m* balls in room A and *n* balls in room B).

	4–0	*4–1*	*4–2*	*5–0*	*5–1*	*5–2*	*6–0*	*6–1*	*6–2*
SP	0.5	1.5	1.1	7.0	21	119	144	1047	(>12 *h*)
ABP	0.2	0.2	0.3	1.9	7.5	23	35	401	5230

Table 2. Planning time (*s*.) for BW problems with four (4-#), five (5-#) and six (6-#) blocks.

```
(define (domain Blocksworld)
   (:ObjectTypes block table - object)
   (:PlaceTypes Stack[object])
   (:action PutOn
      :parameters (x - block  y - object)
      :pre  (Stack(x|_)  Stack(y|_)
      :post (Stack(_|_)  Stack(y|x)
   )
)

(define (problem Sussman)
   (:domain Blocksworld)
   (:Objects A B C - block T - table)
   (:Places s1 s2 s3 - Stack)
   (:init
            s1[T|A|C|_|_]
            s2[T|B|_|_|_]
            s3[T|_|_|_|_] )
   (:goal
           Stack(C|B|A) )
)
```

Figure 5. BW domain description and example of problem instance for the ABP analogical planner (adapted from (Garagnani & Ding, 2003)).

Analysis

Why did ABP invariably perform better than SP, given that they both solved the same problems in the same search space, using the same search algorithm? The answer lies in the two different representations that the planners adopted. In particular, the main factor leading to the gain in performance obtained in these experiments appears to be that analogical planning can exploit the inherent structure of the domain to speed-up the operations of state "look-up" (required to establish the applicability of an operator to a state) and state "update" (carried out as a consequence of the application of an operator).

For example, consider the process of checking whether the BW operator *Move*(*x*,*y*,*z*) – in which a block *x* is moved from the top of object *z* onto another block *y* – is applicable. In the sentential representation, the preconditions *P* could consist of the following set of literals:

$$P = \{\ \texttt{On}(x, z),\ \texttt{Clear}(x),\ \texttt{Clear}(y),\ \neg\texttt{Equal}(x, y)\ \}$$

Assume that the parameters *x*,*y*,*z* are still unbound, and suppose that the checking procedure considers the literals of *P* sequentially, from left to right. The first precondition is compared with the state: if the atom unifies with one of the atoms in the state, parameters *x* and *z* are assigned a value and the process moves on to consider the second literal of the list. However, several unifications of the first atom may have to be discarded before a suitable one is found such that the second atom, `Clear(`x`)`, is also satisfied in the state. If suitable values for x and z are eventually found, the process can move on to the third precondition, `Clear(`y`)`. If the state contains an atom that unifies with `Clear(`y`)` and such that $y \neq x$, the procedure terminates successfully. Otherwise, the algorithm backtracks, and the process is repeated with the next pair of atoms in the state (i.e., the next pair of values for x and z) that satisfy the first two preconditions. In general, the set of possible pairs of such atoms present in the state is a subset of the total possible number of pairs (x,z), i.e., it has cardinality $O(m^2)$, where m is the number of objects (blocks) of the domain. For each pair, a number of instances of `Clear(`y`)` will have to be discarded in order to find one such that $y \neq x$; this will be in the order of $O(m)$, leading to a total number of $O(m^3)$ steps. Hence, in general, it appears that the number of steps required to check the preconditions of an operator is in the order of $O(m^k)$, where k is the total number of parameters that appear in the preconditions.[11]

Let us now consider what happens in the array-based representation. The preconditions P of the *Move*(x,y,z) operator, described at the beginning of this section, consist of the following:

$$P = \{\ stack_1(\ z\,|\,x\,|\,\_\ ),\ \ stack_2(\ y\,|\,\_\ )\ \}$$

As before, suppose that these expressions are considered from left to right. Variable $stack_1$ can be bound to any of the arrays in the state. Once an array has been identified, it is easy to see that the check for the presence of a sequence (such as '$z\,|\,x\,|\,\_$') of k consecutive elements can be carried out in time $O(k\ m)$, where m is the length of the array (i.e., number of blocks of the domain plus one). If this check is not successful, the variable $stack_1$ is assigned to the next possible array in the state, and the process is repeated. Notice that the number of arrays (stacks) present in the state grows at most *linearly* with the number of objects (blocks). The same reasoning can be repeated, of course, for the second precondition.

Similar differences in the efficiency of the state look-up operations between sentential and analogical planning representations should also be expected in the version of BW containing domain axioms. In fact, consider the process of verifying the applicability of the *Unstack*(x,y) operator when axioms (α_1),(α_2) are present. As discussed in the first section of this chapter, the check for the truth of a ground instance of the predicate `Above(`x,y,n`)` requires, in the sentential case, a number of steps polynomial in the number of blocks. On the other hand, the last section illustrated how a graph-based description can encode the BW relations 'on' and 'above' as labelled edges. Hence, the problem of deducing whether a block is above another one becomes one of simply checking the graph for the existence of an edge of the appropriate type between the two nodes containing the blocks. This check can be carried with a number of steps *linear* (or even less) in the total number of blocks.[12]

Now consider the *update* operations caused by the execution of an action. A sentential description of the *Move* action in BW should include, in its effects, the literals `On(`x,y`)`,¬`On(`x,z`)`,¬`Clear(`y`)`. The last effect is a trivial consequence of the action: if block x is on y, then y has something on it. Yet, the sentential model must explicitly include this effect and consider it during the reasoning process. The use of an axiom such as

$$(\neg\,\exists x : \mathtt{On}(x,y)) \rightarrow \mathtt{Clear}(y)$$

would obviate the explicit use of predicate `Clear(`x`)` in the operators, but would not relieve the planner from still having to carry out many trivial calculations, required to take this axiom into account during the search process. In contrast, in the analogical model, this effect is *implicit* in the action of moving a symbol to a new node (or cell): what lies beneath becomes implicitly not clear. The state changes produced by the execution of the analogical operator of Figure 4.(*b*) (or by the array-based version) consist simply of transferring symbol *x* from its original location to its destination. Notice that this operation can be performed in time linear in the number of objects, whereas in case of a sentential model containing domain axioms, updating the ‘above’ relation requires a polynomial number of steps.

In summary, the speed-up achieved by the adoption of setGraphs derives from their ability to carry out more efficient state look-ups and updates. There are two main reasons why these processes are more efficient here than in a sentential representation. The first one follows from the ability of analogical models to *structure* the domain description so that it reflects the inherent spatial (or semantic) structure of the domain. In fact, a domain is often composed of several connected *sub-structures* (e.g., in BW, the stacks) presenting an internal structure simpler than the complete state description; once one of these sub-structures has been identified, a “local” check for the existence of certain conditions or manipulation of elements within it is much simpler and faster than carrying out the same operations on random parts of the global state. In short, the analogical descriptions can be seen as *decomposing* the domain into sub-parts which allow simpler look-up and update operations. The second reason lies in the ability of analogical models to capture *implicitly* the basic, trivial – yet pervasive – physical constraints and properties of a domain and thus relieve the model from having to explicitly include them as additional formulæ or axioms, and take them into account during the reasoning process (see also (Myers & Konolige, 1995)). For example, in the BW domain, the constraint specifying that any block having something put on it becomes “not clear” is *implicit* in the analogical representation: the domain description does not contain *any* explicit formula or element specifying such a constraint.

One question emerging from this study is whether the observed gain in performance is only limited to the so-called *move* domains (domains that involve – or which can be transformed into equivalent ones involving – the movement and manipulation of objects subject to physical and spatial constraints (Hayes & Simon, 1977; Garagnani, 2003), or whether setGraphs, and, more in general, analogical representations can also be applied successfully to other types of planning problems. In order to address this question, the next section provides a more general analysis of the characteristics, advantages and limitations of analogical descriptions with respect to sentential ones. The conclusions drawn from this analysis will lead to the second part of this chapter, in which a way to exploit the advantages of both representations within a single framework is proposed.

CHARACTERISING ANALOGICAL MODELS

The need for formalisms to support common-sense reasoning more efficiently than the traditional sentential (or *Fregean* (Kulpa, 1994)) representations has recently lead to a resurgence of interest in “non-linguistic” descriptions, also referred to as *diagrammatic* (Larkin & Simon, 1987), *analogical* (Sloman 1975; Dretske, 1981), *homomorphic* (Barwise & Etchemendy, 1995), *direct* (Levesque, 1986) and *model-based* (Barr & Feigembaum, 1981; Halpern & Vardi, 1991). Let us analyse the general characteristics of these representations and the elements that allow discriminating them from sentential ones.

Analogical *vs.* Sentential

The feature that most clearly distinguishes analogical models from sentential ones is the relation existing between the *syntax* of the formulæ of the language and the *semantic* structure of the represented domain. In analogical representations, the *syntax* of the language structures mirrors, for the relevant aspects, the semantics (relations and properties) of the domain represented (Barr & Feigembaum, 1981, pp.200–206). In other words, the world is modelled using descriptions that are *structurally* similar to the object, situation or event represented. In contrast, the syntax used for the formulæ of a sentential representation has no particular relevance, and its specific structure has no bearing to the specific structure of the represented domain (Kulpa, 94). Barwise and Etchemendy (1995) concisely characterise this difference:

"[…] *with homomorphic representations the mapping ϕ between syntactic structure (that is, the structure of the representation itself) and semantic structure (that is, the structure of the object, situation or event represented) is highly structure preserving, whereas the corresponding mapping in the case of linguistic representations is anything but structure preserving*." (p. 214)

According to Palmer's (1978) characterisation, the relations between elements of analogical structures and the corresponding represented relations of the domain have the same algebraic structure (i.e., they are *naturally isomorphic*). In addition, unlike in sentential descriptions, the relevant relations between objects of the domain do not need to be *explicitly* declared in the domain model and appear as "pointable" elements of the description.

In order to clarify the previous definitions, let us compare the array-based analogical representation of BW presented earlier against its sentential version. In the latter, the relevant relations ('on top of ', 'above') between objects are specified using relational instances, which are pointable elements of the state (*viz.*, formulæ, like `On(A,B)` and `Above(C,D,1)`). In addition, the *properties* of these relations and their interactions are imposed on the model by logical expressions – e.g., axioms (α_1),(α_2) – that extend (or restrict) the legal set of instances of a certain predicate. In contrast, in the analogical representation the objects and the relevant spatial relationships that exist between them are not described as sets of relational instances, but modelled using appropriate data structures (arrays). The *syntax* of such data structures mirrors, for the relevant aspects, the semantics of the domain. In fact, the formulæ used to describe a BW state have the following syntax:

$$name\ [\ arg_1 \mid arg_2 \mid \dots \mid arg_n]$$

This notation clearly reflects the semantics of BW: the first argument of the formula (arg_1) always represents the "table" object; the second, the lowest block of a stack, lying on the table; two consecutive symbols arg_k, arg_{k+1} indicate that block arg_{k+1} is on block arg_k, and the rightmost non-empty symbol (i.e., different from '_') of the formula denotes a clear block. In contrast, the formulæ used in the sentential representation adopt the following syntax:

$$predicate(arg_1, arg_2, \dots, arg_n)$$

This syntax has no direct relation with the structure and properties of the BW domain. The specific *structure* of this formula (i.e., the number and order of its arguments) has no particular connotation, valid for all of the formulæ of the language. For example, unlike in the array-based representation, there is no specific role associated to the first argument of a formula, valid for *all* the predicates of the language. Moreover, consider the spatial relation 'above', represented in the array model by the relation 'to the right of ', defined on the symbols of the array[13]. First of all, this relation is not represented explicitly, as a pointable element of the state. Secondly, the transitive property of the relation, imposed on the sentential predicate `Above(`x,y,n`)` by axioms (α_1),(α_2), is an *implicit* property of the relation 'to the right of ', and does not need to be explicitly imposed on the model, included in the description and accounted for during the reasoning process. In other words, the transitivity of the relation is an *emergent property* of the representation (Koedinger, 1992); to use Palmer's terms, the relation 'to the right of ' in the array model and the corresponding spatial relation 'above' in the real domain are naturally isomorphic (Palmer, 1978).

Advantages and Limitations of Analogical Models

Myers & Konolige (1995) observed that one of the key features of analogical representations is their "capacity to implicitly embody constraints that other representations must make explicit" (p. 275). The analysis of the experimental results obtained with the array-based planner illustrated how the ability of analogical models to implicitly encode the basic physical properties and constraints of a domain and to reflect its internal (topological or semantic) structure can lead to better planning performance, particularly when a domain can be decomposed (according to its spatial or semantic structure) into smaller – possibly linearly structured – parts that enable efficient, "localised" condition-checks and element manipulations. These features significantly reduce the computational load involved in state update and look-up operations, which represent a substantial part of the overall planning and reasoning process. Interestingly, the need to perform these operations is closely linked to the presence of the well known *frame* and ramification problems.

The problem of ramification of the effects of action (directly associated to the presence of domain axioms – see the earlier section "Domain Axioms and the Ramification Problem") is just another facet of the frame problem (McCarthy & Hayes, 1969): while the former is caused by the need to reason about the properties of the world that *change* as a consequence of an action, the latter concerns reasoning about the aspects that do *not* change. The frame problem is still regarded as presenting a major difficulty for reasoning about action (Shanahan, 1997). In the analogical formalisms presented earlier, the frame problem is addressed exactly like in sentential ones, i.e., by requiring that an operator explicitly contains only the changes resulting from the execution of the represented action, and assuming that all the remaining aspects of the state are left unchanged (Lifschitz, 1990). Assuming such "default persistence" provides only a simplistic solution to the frame problem, and requires an operator to specify *all* the possible consequences that the execution of the corresponding action has on the state – in other words, it leads to the ramification problem.

The adoption of analogical representations has a lessening effect on the frame/ramification problem. Sentential planning languages are generally more flexible and expressive than analogical ones; however, because of their "unconstrained" nature, they require *all* the properties and constraints of the domain – even the most trivial – to be represented "extrinsecally" (Palmer, 1978) in the domain, i.e., to be *explicitly* imposed on the model using "pointable" elements (formulæ, axioms of the language) which have to be taken into account during the reasoning process. Analogical representations can make some of such constraints (axioms) *implicit* in the model; in addition, they may allow the domain to be decomposed in simple sub-structures that enable *localised* (as opposed to global, "ramificated") state look-ups and updates. This significantly reduces the number of deduction steps required, and, hence, eases the ramification problem (see also (Lindsay, 1995)). In particular, as discussed earlier in the analysis of the experimental results, the use of sentential description leads, in general, to a polynomial number of operations required for state look-up and update operations. This can be reduced to just *linear* complexity through the adoption of analogical models.

There is a second way in which analogical models may be able to reduce the negative effects of the frame problem. In sentential planning languages, the assumption that nothing else changes apart from the effects explicitly specified by the action leads to the formulation of rather complex conditions for determining when two actions can be executed *simultaneously* (e.g., see the conditions for mutually exclusive actions in PDDL2.1 (Fox & Long, 2003)). Indeed, a significant amount of effort is spent by Graphplan (Blum & Furst, 1997) and similar systems to calculate *all* such pairs of "mut-ex" operators. In contrast, analogical descriptions appear to allow a much simpler check: two operators can be executed simultaneously if they act upon parts of the analogical model that are *disjoint*. This condition is not strictly necessary, but it is sufficient, and it suggests that analogical descriptions may lead to further speed-ups if used in conjunction with Graphplan-based algorithms.

One of the main problems of purely analogical representations, however, is to find a sufficiently general model that can represent *all* complex aspects of the real world and still allow efficient descriptions. Due to the implicit, unalterable structure of the relations that they use, analogical representations are usually criticised for their limited expressiveness and tendency to be domain (or, at best, "generic-domain") specific. For example, the setGraph representation proposed in the first section appears to be suitable for representing generic *move* domains, involving the manipulation of objects in topological or structured spaces. However, can setGraphs also be used to represent other types of domains, involving, for example, no movement at all? It is not difficult to show that setGraphs can encode *any* domain such that the current state can be described as a finite set of objects O=$\{x_1,\ldots,x_m\}$, each being in one of a *finite* number of possible states.[14] Indeed, the theoretical results presented in the next section show how setGraphs can be extended so as to become expressively equivalent to a propositional planning language. It remains to be seen whether such formalism is generally more efficient than other, sentential or state-variable based ones (e.g., (Cesta & Oddi, 1996)).

Indeed, in spite of its expressiveness, it appears unlikely that even an extended setGraph formalism would be able to describe *all* problems more efficiently than any other sentential representation. A similar objection, however, applies equally well to purely sentential planning formalisms. In short, it seems that no single, purely analogical or purely sentential representation paradigm exists that can be used to describe *all* possible problems more efficiently than any other: the complexity of real-world applications requires from a language a "mixture" of different capabilities that analogical or sentential models alone cannot offer.

In view of this, the knowledge representation community has been investigating the use of *heterogeneous* (or *hybrid*) models (Barwise & Etchemendy, 1995, 1998; Myers & Konolige, 1995; Swoboda & Allwein, 2002), in which different types of representations are integrated and used by the system to construct threads of proof which cross the boundaries of sentential and non-sentential paradigms of representation. The advantage of a hybrid system with respect to a purely sentential or analogical one is that it allows to encode and reason about different aspects of the world using the *most appropriate* (i.e., efficient) representation for each aspect. The next section describes a framework for hybrid planning, in which domain descriptions containing qualitative, quantitative, sentential and analogical features can be integrated and used interchangeably. The analogical model adopted extends the setGraph formalism introduced before, making it expressively *equivalent* to the sentential model of action adopted. The result is a powerful, heterogeneous planning representation that combines the strengths and overcomes the limitations of the two paradigms on which it relies.

A FRAMEWORK FOR HYBRID PLANNING

This section proposes a model for integrating sentential planning representations with analogical ones into a single heterogeneous formalism. The contents of this section, largely based on the ideas described in (Garagnani, 2004), are divided into four parts. In the first part, the setGraph formalism introduced only intuitively in the previous sections is formulated in more rigorous terms and extended into a more expressive representation, allowing types and numeric quantities. The second part briefly describes the sentential model chosen, based on the current standard planning domain description language, PDDL2.1 (Fox & Long, 2003). The third part proposes a model for hybrid planning that integrates the two representations, and illustrates the approach through an example. The last subsection presents a general theory that allows the identification of the conditions for the *soundness* of hybrid planning models.

The Analogical Model: Extending SetGraphs

We begin by extending and recasting in more formal and rigorous terms the setGraph model proposed earlier. The model is augmented so as to allow (1) types and numeric values (hence, attributes with *infinite* domains), and (2) actions involving non-conservative changes (additions and removal of elements to and from a setGraph) and numeric updates. The extension of setGraphs with numeric quantities can be seen as the first step towards the "hybridisation" of the model, completed later on by its integration with a sentential language.

Typed and numeric setGraphs

In order to formally define a setGraph, let us introduce the *collection* construct. A collection is a data structure identical to a *list*, except that the order of the elements is unimportant. Equivalently, a collection can be seen as a set in which multiple occurrences of the same element are permitted. Notice that the multiple instances of an element should be thought of as distinct elements of the structure. For example, C={1,1,0,0,0} denotes a collection of integers containing two occurrences of the number 1 and three occurrences of the number 0. Since the order is unimportant, any permutation of the elements of C is equivalent to the same collection. Hence, C={1,0,1,0,0}={1,0,0,1,0}={1,0,0,0,1}={0,1,0,1,0}=… etc.

The empty collection is denoted as { }. We adopt the notation "$x \in$ C" and say that *x* is *contained* in C to indicate that element *x* appears (occurs) at least once in collection C.

The definition of a setGraph is based on that of *nodeSet*, specified recursively as follows:

Definition 1 (nodeSet, node, place) *Let W be a set of strings (language). A* nodeSet *is either:*

- *a string $w \in W$ (in which case, the nodeSet is a* node*), or*
- *a finite collection of nodeSets (in which case, the nodeSet is a* place*).*

A node is a string of the language *W*. A place is a "container" for both nodes and places. Nodes and places are nodeSets. In short, nodeSets are data structures consisting of multi-nested sets of strings with multiply occurring elements and no limit on the level of nesting. For example, consider a language *W*={Ab} with one string only; each of the following represents a nodeSet (the notation {*x*,*y*,…,*z*} indicates a place containing nodeSets *x, y*,…, *z*):

Ab (4.1)
{ } (4.2)
{Ab, {Ab}, {{Ab}} } (4.3)
{{Ab}, {{ Ab, Ab }}, {{ },{ }}, {Ab}, {{ },{ }}, {Ab} } (4.4)

Formula (4.1) represents a node; (4.2) represents an empty place, and (4.3) a place containing one node and two places, of which one contains a node and the other one a place.

In the nodeSet notation adopted, places can be associated to *labels* (strings) which can then be used to refer to the elements of a nodeSet structure. For example, if *p, q, r* and *s* are labels, the nodeSet identified by (4.3) could also be specified by the expression (4.5) below:

$$p\{\ \mathrm{Ab},\ r\{\mathrm{Ab}\}, q\{\ s\{\mathrm{Ab}\}\}\ \} \qquad (4.5)$$

Any reference to place q identifies nodeSet $q\{s\{\mathrm{Ab}\}\}$, *viz.*, $\{\{\mathrm{Ab}\}\}$. Notice that place labels are not required to be distinct (as explained below, this is useful when *types* are introduced).[15]

Given a nodeSet N, $\wp(N)$ is defined as the *collection* of *all* the nodeSets occurring in N (including N itself). For example, consider a language $W=\{\mathrm{A,B,C}\}$. Let N_1 be the nodeSet identified by expression $\mathrm{P}_0\{\mathrm{A}, \mathrm{P}_1\{\mathrm{B}\}, \mathrm{P}_2\{\ \mathrm{P}_3\{\mathrm{C}\}\}\}$. Then, $\wp(N_1)= \{\mathrm{A, B, C, P_0, P_1, P_2, P_3}\}$.

Definition 2 (setGraph) *A* setGraph *is a pair* $\langle N,E\rangle$, *where* N *is a nodeSet and* $E=\{E_1,\ldots,E_k\}$ *is a finite set of binary relations* E_i *, such that* $E_i \subseteq \wp(N)\times\wp(N)$.

If E contains only one relation E_i, we write simply $\langle N, E_i\rangle$. For example, let N_1 be the nodeSet specified as $N_1=\{\mathrm{A},\{\mathrm{B}\},\{\{\mathrm{C}\}\}\}$. The pair $\alpha=\langle N_1,E_1\rangle$ is a setGraph, where

$$E_1=\{(\mathrm{C}, \mathrm{B}),\ (\{\mathrm{B}\}, \{\{\mathrm{C}\}\}),\ (\{\mathrm{A},\{\mathrm{B}\},\{\{\mathrm{C}\}\}\}, \mathrm{A})\ \}$$

The instances of the binary relations E_i – pairs of elements of $\wp(N)$ – are the *edges* of the setGraph. Notice that if N_1 is specified using the notation $N_1=\mathrm{P}_0\{\mathrm{A}, \mathrm{P}_1\{\mathrm{B}\}, \mathrm{P}_2\{\mathrm{P}_3\{\mathrm{C}\}\}\}$, as in the previous example, then E_1 can be written also as $\{(\mathrm{C}, \mathrm{B}),\ (\mathrm{P}_1, \mathrm{P}_2),\ (\mathrm{P}_0, \mathrm{A})\}$.

SetGraph structures have a direct graphical interpretation. Figure 6.(*a*) contains a graphical representation of the setGraph $\alpha=\langle N_1,E_1\rangle$, where places are depicted as ovals, nodes as the corresponding strings of the language, and edges as labelled arcs. All (and only) the nodeSets that are contained in a place appear within the perimeter of the corresponding oval.

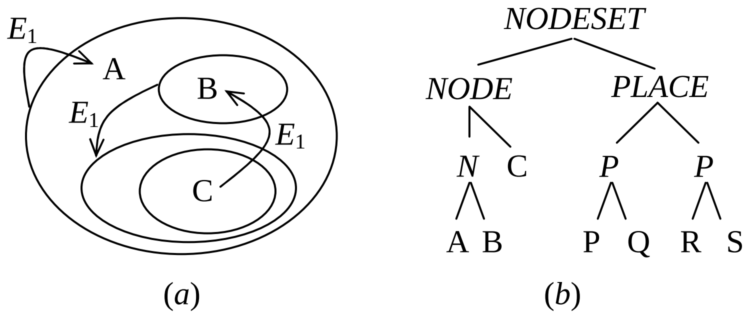


Figure 6. (*a*) Graphical representation of setGraph $\alpha = \langle N_1,E_1\rangle$;
(*b*) associated type hierarchy

In a setGraph, the relations E_i denote different *types* of edges, represented in Figure 6.(*a*) as arc labels. Similarly, nodeSets can also be required to be of specific types (or *sorts*). Figure 6.(*b*) contains a tree of labels representing an "IS-A" hierarchy of types. The root of the nodeSet hierarchy is always the type *NODESET*. The leaves of the tree are called *instances*. Each node of the tree identifies a type. Each type t represents the set of instances of the sub-tree having t as root (e.g., *NODE*=$\{\mathrm{A,B,C}\}$). Types *NODE* and *PLACE* are always the only sub-types of *NODESET*. If a setGraph G is associated to a type hierarchy (as in Figure 6), G is said to be *typed*. In a typed setGraph, the instances of the type *NODE* form the language W. In what follows, all setGraphs are assumed to be typed, unless otherwise specified.

The introduction of types allows to “characterise” and differentiate the nodeSets of a setGraph. Different types of places (and nodes) may have different properties and behaviour, which are inherited by all sub-types and instances (see Example 5.1 below). In a typed setGraph, the type of a node is unambiguously identified, since nodes are strings, instances of *NODE*. In order to specify the type of a place, we adopt the same labelling notation introduced earlier for identifying places: the type of a place is specified by associating the place to a label, instance of *PLACE*. To avoid ambiguities, places of the same type can be discriminated using distinct *variable names* of the same type (see below).

The use of types (and typed variables) in setGraph descriptions yields *generalised* setGraphs. A generalised setGraph is obtained from a setGraph by replacing one of the nodeSets with one of its *super-types* (or with a variable of that type). A generalised setGraph denotes the *set* of setGraph descriptions that can be obtained from it by replacing all types (and variables) with appropriate instances. For example, consider Figure 6.(*a*). The four places of the setGraph are not associated to any label. Unless otherwise specified, all places of a setGraph description are assumed to be of type *PLACE*. Hence, Figure 6.(*a*) is a generalised setGraph, representing the set of setGraphs that can be obtained by labelling each place with any of {P,Q,R,S}=*PLACE*. To use a textual notation, Figure 6.(*a*) is equivalent, for example, to the parameterised setGraph description $\langle N_2, E_2\rangle$, where

$$\begin{aligned} N_2 &= x\{\mathrm{A},\ y\{\mathrm{B}\},\ w\{\ \mathit{PLACE}\{\mathrm{C}\}\}\} \\ E_2 &= \{(\mathrm{C},\mathrm{B}),\ (y,\ w),\ \ (x,\mathrm{A})\} \end{aligned}$$

and where variables *w*, *x*, *y*, (called the *parameters* of the setGraph) are of type *PLACE*.

Notice that in parameterised setGraphs a variable name may appear only once to identify a node or a place, whereas the same *type* may be used to label different nodes (or places). Given a type hierarchy, a setGraph description containing only instances of the hierarchy (i.e., no types or variables) and identifying only one – typed – setGraph is said to be *ground*.

Actions with numeric and non-conservative effects

In addition to the representation of the (initial) world state (consisting of a ground setGraph), a planner must also be provided with a specification of how states are changed by actions. In order to allow specifying operators with numeric preconditions and effects, the notation for analogical operators adopted earlier needs to be extended. In addition, the setGraph formalism proposed there was limited to actions consisting simply of moving nodeSets from one place to another; this model is augmented here to allow actions that add elements to and remove elements from a setGraph, enabling a state to undergo “non-conservative” changes.

Numeric quantities are represented in setGraphs as *numeric nodes*. A numeric node is a string of *W* of form “*n.m*” or “*n*” (possibly preceded by ±), or the string “⊥”. The symbols *n*, *m* denote sequences of digits in {0,1,…,9}. The node “⊥” is used to represent numeric attributes with undefined values. The *value* of a numeric node (string) is calculated using a function $val{:}W\rightarrow\Re_\perp$, where $\Re_\perp=\Re\cup\{\perp\}$ and $\Re$ is the set of reals. In particular, *val*(*w*) is the (float or integer) number represented by *w* if *w* has form “*n.n*” or “*n*”, ⊥ otherwise. The function $str{:}\Re_\perp\rightarrow W$ returns the inverse of *val*, i.e., $str = val^{-1}$ (e.g., *str*(–1.75) = “–1.75”).

The possible setGraph transformations considered here are: (*i*) *addition* or *removal* of an element, (*ii*) *movement* of a nodeSet, and (*iii*) *re-assignment* (or *update*) of a numeric node.

The movement and removal of elements in a setGraph is based on the following general rules: (1) if a node is moved (removed), all edges linked to it move (are removed) with it; (2) if a place is moved (removed), all the elements contained in it and all edges linked to it move (are removed) with it. Any element not moved, removed or updated is left *unaltered*. In addition, let $x \in \Re_\perp$ be the value *val*(w) of a numeric node w, and let $n \in \Re_\perp$. The possible updates of a numeric node w are: (a) *Assign* ($x' := n$); (b) *Increase* ($x' := x + n$); (c) *Decrease* ($x' := x - n$); (d) *Scale-up* ($x' := x * n$), and (e) *Scale-down* ($x' := x/n$). The result, x', is $\perp$ if one of the operands is $\perp$. The application of one of these updates to the numeric node (string) w causes w to be transformed into the string *str*(x'). (Notice that *str*($\perp$) = "$\perp$").

As usual, the domain-specific legal transformations of a state (setGraph) are defined through a set of parameterised operators. An operator $P \Rightarrow E$ consists of preconditions P (specifying the situation required to hold in the state before the action is executed) and effects E (describing the situation of the state after). However, preconditions and effects contain here two separate parts, analogical and numerical. The analogical preconditions and analogical effects are lists of parameterised setGraphs. The numerical preconditions consist of a set of *comparisons* ($<, >, =, \neq, \leq, \geq$) between pairs of numerical expressions,[16] while the numerical effects consist of a set of *update* operations of the kind (a)–(e) listed earlier.[17]

Example 5.1 – Consider a simple Ferry domain, consisting of two ports (Port_1 and Port_2), a ferry boat, and a number of cars. The ferry can sail between the two ports, carrying a limited number of cars. The cars can board and debark the ferry at either of the two ports. The problem is to find a plan (involving the least number of ferry trips and least number of boarding and debarking operations) that transforms a given initial state into one in which each car has reached a specific port. The (ground) setGraph represented in Figure 7.(a) encodes an example of initial state for a Ferry problem with three cars (A,B,C) and a ferry F_1. Figure 7.(b) contains the associated type hierarchy.

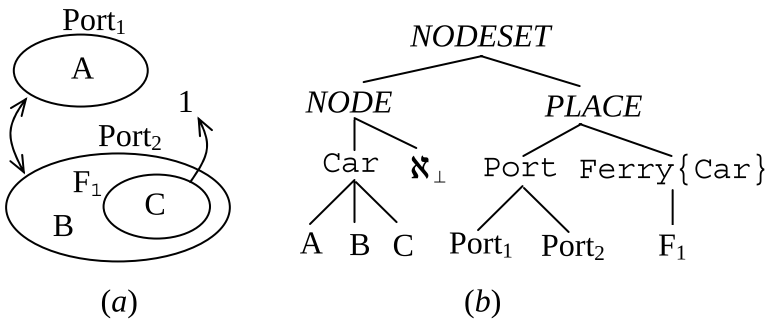


Figure 7. Ferry domain: (a) initial state setGraph; (b) type hierarchy

The setGraph representation of Figure 7.(a) omits arc labels (all edges are of the same type). The bi-directional arc between Port_1 and Port_2 denotes the presence of two symmetric edges connecting the two locations. The arc linking the ferry (place F_1) to node 1 denotes an edge associating this nodeSet to the number of nodes (cars) that it contains. In order to allow the representation of numeric quantities, the language W is extended by adding $\aleph_\perp$ to the type hierarchy as a subtype of *NODE* (see Figure 7.(b)). The type $\aleph_\perp$ can be thought of as having instances "0", "1", "2",…, i.e., the infinite set of strings representing all the natural numbers. The special string "$\perp$" is also an instance of $\aleph_\perp$, representing "undefined" numeric values. Notice that the type hierarchy restricts all places of type `Ferry` to contain only elements of type `Car` (by default, all places are allowed to contain any instance of the *NODESET* type). This property is inherited by all instances of `Ferry` (here, only F_1).

Figure 8 contains the graphical representation of the analogical operator *Board*(x,y,z) for the Ferry domain, encoding the boarding of a car y on a ferry z. The analogical precondition and effect lists of this operator consist of only one parameterised setGraph. The numerical parts constrain and update, respectively, the value of the numeric node x. The applicability of the operator (see below) is subject to *y* and *z* being at the same port and to *x* being less than three. Notice that the fact that place *z* in the preconditions *P* does not contain any element should not be interpreted as requiring it to be empty (this will follow from Definition 3 and method (α), presented below).

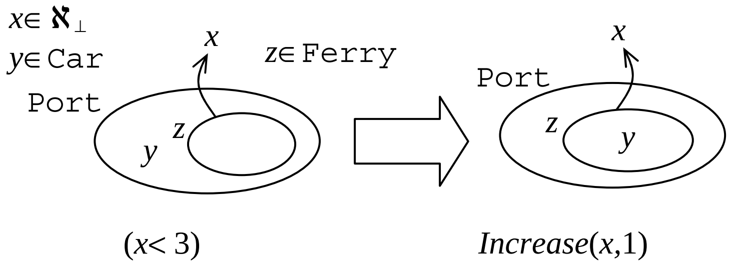


Figure 8. *Board*(*x*,*y*,*z*) operator for the Ferry domain: preconditions *P* (left) and effects *E* (right)

The *Debark* operator will consist essentially of the "reverse" version of the *Board* operator, although *x* will not need to be constrained, but *decreased* by 1.

Having generally illustrated the syntax of setGraph operators, let us now specify their semantics. The semantics of action is specified by providing an algorithmic definition of the following: (α) a method to check whether an operator is applicable to a given state *s*; (β) a method for calculating the state resulting from the application of an operator to a state *s*. These methods (detailed below) make use of the definition of *satisfaction*, specifying the conditions for a parameterised setGraph *T* to "match" a setGraph *G*. Intuitively, *T* is satisfied in (or matches) *G* iff there exists a substitution of all the variables and types of *T* with appropriate instances such that *T* can be made "coincide" with *G* (or with a subpart of it):

Definition 3 (Satisfaction) *Given a parameterised setGraph T={N,E} (with associated type hierarchy) and a ground setGraph G, T is* satisfied *in G* iff *there exist a substitution θ of each parameter (variable) of T with an instance of the appropriate type, and a 1-1 function $\sigma: T \to G$ mapping elements of T to elements of G, such that, if N_θ is nodeSet N after the application of substitution θ, the following conditions are all true:*

- *for all nodes $x \in N_\theta$, either $x=\sigma(x)$, or $\sigma(x)$ is an instance of type x*
- *for all places $y \in N_\theta$, $\sigma(y)$ is an instance of the type of y*
- *for all pairs (x,y) such that $x,y \in \wp(N)$, if $x \in y$ then $\sigma(x) \in \sigma(y)$*
- *for all edges $e=(x,y) \in E$, $\sigma(e) = (\sigma(x), \sigma(y))$*

The first two conditions require that each nodeSet of *T* is either equal to, or a super-type of, the corresponding image in *G*; the third condition requires that any relation of containment between nodeSets of *T* is reflected by containment between the corresponding images in *G*; the last condition requires that if two nodeSets are linked by an edge in *T*, the corresponding images is linked by the image of the edge in *G*.

The definition of satisfaction is used for detailing methods (α) and (β), mentioned earlier:

(α) *An operator $P \Rightarrow E$ is applicable in a state (setGraph) s iff (1) all the parameterised setGraphs of P are satisfied in s (using binding σ and a* single *substitution θ replacing equal variables with equal instances), and (2) if every occurrence of each numeric* *variable x in the numeric part of P is replaced with the value val(σ(x)), all the numeric comparisons in P are true;*

(β) *If operator O is applicable in state s, the result of applying O to s is the new setGraph obtained from s by (1) carrying out – on the corresponding elements of s identified through binding σ – the changes required to transform each of the setGraphs in the preconditions P into the (respective) setGraph in the effects E, and (2) for each update operation of E, updating the numeric nodes with the result of the respective operation.*[18]

Example 5.2 – Consider the Ferry domain of Example 5.1. Given the type hierarchy of Figure 7.(*b*), the preconditions *P* of the *Board*(*x*,*y*,*z*) operator of Figure 8 are satisfied in the setGraph specified by Figure 7.(*a*). In fact, let $\theta = (x/1,\ y/\mathrm{B},\ z/\mathrm{F}_1)$ be a substitution of parameters with instances of appropriate type (notice that θ is legal, as $1 \in \aleph_\perp$, B∈ `Car` and $\mathrm{F}_1 \in$ `Ferry`). In addition, let σ map the nodeSets of the analogical preconditions of Figure 8 to nodeSets of the setGraph specified in Figure 7.(a) as follows: nodes *x*,*y* to nodes 1,B (respectively), the place labelled *z* to the place labelled F_1, edge (*z*,*x*) to edge $(\mathrm{F}_1,1)$, and the place labelled `Port` to the place labelled Port_2. If P_θ is the setGraph obtained by applying substitution θ to the analogical part of preconditions *P*, then, for all nodes $x \in P_\theta$, $x=\sigma(x)$ (this is obvious), and for all places of P_θ, the images are instances of their respective types (in fact, consider place *z*∈ `Ferry`: the image is place F_1, instance of `Ferry`; consider place labelled `Port`, of type `Port`: the image is place Port_2, an instance of `Port`). In addition, it is easy to see that containment between nodeSets of *P* is reflected by containment between the corresponding images, and the only edge $e=(z,x)$ in *P* is such that $\sigma(e)=(\mathrm{F}_1,1)=(\sigma(z), \sigma(x))$, as required by Definition 3. Finally, in the numeric part of *P*, if the occurrence of variable *x* is replaced with value $val(\sigma(x))=1$, the comparison ($x< 3$) is satisfied. Therefore, *Board*(*x*,*y*,*z*) is applicable to the ground setGraph depicted in Figure 7.(*a*). The application of the operator would transform the setGraph into one in which car (node) B is inside the ferry (place F_1) and the numeric node “1” has become “2”, as expected.

Notice that the use of a graphical representation for specifying analogical operators is due to purely explanatory reasons. Analogical operators can also be specified textually, using a notation analogous to the one adopted for array-based analogical planning (see the third section of this chapter). For example, consider the *sail* action of the Ferry domain, consisting of the transfer of the ferry (and of its contents) from one port to the other. The analogical, setGraph operator representing this action could be specified textually as follows:

Sail (*x*, *y*, *z*) *-- Moves ferry x from port y to port z*
Parameters: *x* – `Ferry`; *y*, *z* – `Port`
P: $\langle \{\ z\{x\{\ \}\}, y\{\ \}\}\ \}, \{(z, y)\}\ \rangle$
E: $\langle \{\ z\{\ \}, y\{\ x\{\ \}\}\ \}, \{(z, y)\}\ \rangle$

As mentioned before, specifying empty places in the preconditions (e.g., *x*,*y*) is not equivalent to requiring that such places be empty. Indeed, in order to express the condition of “emptiness” of a place, a specific notation should be adopted.

The Sentential Domain Description Language

Having reformulated and extended the analogical representation to allow types and numeric quantities, let us briefly describe the sentential model adopted, which is *expressively* equivalent to the setGraph model presented in the previous section (see Theorem 1 below).

The sentential representation language adopted is based on PDDL2.1 (Fox & Long, 2003). The semantics of PDDL2.1 builds on and extend the original core of Lifschitz' STRIPS semantics (Lifschitz, 1990) to handle durative actions, numeric and conditional effects. The action description proposed here, however, is a simplified version of PDDL2.1, and is better thought of as an extension of STRIPS to numbers and functor symbols.

As in PDDL2.1, the world state description is composed here of two separate parts, a *logical* (STRIPS-like) state and a *numeric* state. While the logical state S is a set of ground atomic formulæ (and the truth of an atom p depends on whether $p \in S$), the numeric state consists of a finite vector of real numbers, containing all the current values of the possible *primitive numeric expressions* (PNEs) of the problem. A PNE is a formula $f(c_1,\ldots,c_n)$, where $c_i \in C$ is a symbol representing an object, and f is a functor symbol representing a function $f{:}C^n \rightarrow \Re$ (see example below – a more precise definition is given later on in this Section). The truth of a comparison ($<$, $>$, $=$, $\neq$, $\leq$, $\geq$) between two numeric expressions (containing PNEs and real numbers) in a state S is obtained by replacing each occurrence of each PNE in the comparison with the corresponding numeric value, taken from the current vector of S .

According to the above, a sentential operator $P \Rightarrow E$ specifies a transformation of a state-pair s=(*logical, numeric*) into a new state-pair s'. In the notation considered here, the preconditions P contain simply a set of literals and comparisons between pairs of numeric expressions. The effects E are a set of literals and update operations of the form *Op*(*w, expr*), where *Op* is one of the five update operators *Assign, Increase, Decrease, Scale-up* and *Scale-down* used earlier for the setGraph operators, w is a PNE, and *expr* is a numeric expression (combining PNEs and/or real numbers with operators $+$, $-$, $*$, $/$). As usual, operators are parameterised, i.e., the literals in P and E can contain typed variables.

For example, consider the *Board* action for the Ferry domain (Example 5.1). This action, represented using setGraphs in Figure 8, could be encoded in the sentential model as follows:

Board(*x*,*y*,*z*)	-- *Boards car x (currently at port z) onto ferry y (also at port z)*
Parameters:	*x* – `Car`; *y* – `Ferry`; *z* – `Port`
P:	{ `At(`*x*, *z*`)`, `At(`*y*, *z*`)`, `<(tot_cars(`*y*`), 3)` }
E:	{ `OnBoard(`*x*, *y*`)`, `¬At(`*x*, *z*`)`, `Increase(tot_cars(`*y*`), 1)` }

As from the hierarchy of Figure 7.(*b*), *Car*={A,B,C}, *Ferry*={F_1}, *Port*={Port_1, Port_2}. The symbol `tot_cars` must be declared in the domain description as functor of one argument; the numeric value returned by `tot_cars(`*x*`)` is the number of cars currently on board of ferry *x* (for a detailed description of the semantics of this language, see (Fox & Long, 2003)).

Notice that any PDDL2.1 "level 2" (i.e., without durative actions) operator can be compiled into an equivalent set of ground operators of the above form (Fox & Long, 2003). In view of this, we refer to the sentential formalism described above as to PDDL2.1-*lev2**.

Theorem 1 (Equivalence) *Any setGraph encoding of a planning domain can be transformed into an equivalent sentential (PDDL2.1-lev2*) description, and vice versa.*

Proof – Consider the first part of the theorem. We first show how to transform every ground setGraph into a sentential state s=(*logical*, *numeric*). We then argue that, within such encoding, any setGraph operator can be transformed into an equivalent sentential operator.

By definition, a setGraph is a pair $\langle N,E\rangle$, where N is a nodeSet and E a set of binary relations on $\wp(N)$. Let each nodeSet $x \in \wp(N)$ of N (including numeric nodes) be associated to a unique label l_x that identifies it. The setGraph data structure can then be entirely described using two predicates, $link(e,l_x,l_y)$ and $in(l_x,l_y)$, expressing, respectively, the presence of edge $(x,y)\in e$ (where $e\in E$) and that nodeSet x is an element of y (e.g., see state I in the second section of this chapter). In addition, for each numeric node x, the label l_x can be used as 0-placed function and assigned the value of x through the vector of the numeric part of the sentential state. Given this encoding, every analogical transformation of a setGraph G into G' can be "simulated" in the sentential representation by adding or removing the appropriate atoms to/from the current logical state L, so that L' will represent G'. The update of a numeric node is encoded as the update of the corresponding value in the PNE vector.

Consider the second part of the theorem. We first show how to transform every sentential state s=(*logical*, *numeric*) into a corresponding setGraph, and then how any sentential operator can be encoded by an equivalent setGraph operator in this representation.

Every state $s=(L,R)$ consists of a finite set L of ground atoms $p(x_1,\ldots x_n)$ and a finite vector R of numeric values y_j, each one representing the value in s of the j-th primitive numeric expression $f(x_1,\ldots x_m)$ (where $x_i\in C$, and C is the set of constant symbols representing the entities of the domain). Let G be a setGraph containing the following: (1) three places, labelled *Pred*, *Obj* and *Funct*; (2) a node "c" in *Obj* for each symbol $c\in C$; (3) a node "p" in *Pred* and a set of labelled edges $\{e_1(p, x_1),\ldots e_n(p, x_n)\}$ for each atom $p(x_1,\ldots,x_n)$ in L; and (4) a node "f" for each functor symbol f and a set of nodes $\{x_1,\ldots x_m, str(y_j)\}$ in *Funct* linked by a set of edges $\{(f,x_1),(x_1,x_2),\ldots,(x_m, str(y_j))\}$ for each value y_j in R. Then, the truth of an atom $p(x_1,\ldots x_n)$ can be determined by checking if the setGraph $\langle\{Pred\{p,x_1,\ldots,x_n\}\}, \{e_1(p,x_1),\ldots e_n(p, x_n)\}\rangle$ is satisfied in G. Moreover, the value of the j-th PNE is identified by the value to which the variable $w\in\mathfrak{R}_\perp$ has to be bound for the parameterised setGraph $\langle\{f, x_1,\ldots, x_n, w\},\{(f,x_1),(x_1,x_2),\ldots,(x_m, w)\}\rangle$ to be satisfied in G. For example, Figure 9 depicts the setGraph obtained from a sentential description of the Ferry state of Figure 7.(*a*).

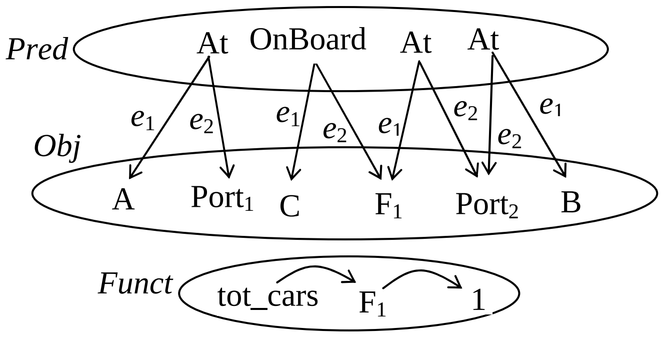


Figure 9. Theorem 1: setGraph equivalent of a PDDL2.1-*lev2**
sentential state (Ferry domain).

Given the above encoding[19], every sentential operator can be transformed into an equivalent setGraph operator as follows: each addition (removal) of an atom $p(x_1,\ldots x_n)$ to (from) the logical state L corresponds to the addition (removal) of the corresponding node "p" and associated edges to (from) place *Pred*. Similarly, each update of a PNE $f(x_1,\ldots x_m)$ in R is encoded through the update of the numeric node w at the end of the "chain" (f,x_1), (x_1, x_2), $\ldots,(x_m, w)$. For example, Figure 10 depicts the parameterised setGraph operator obtained from the sentential version of *Board*(x,y,z), presented earlier in this section.[20] Q.E.D.

Notice that the transformation of sentential descriptions into setGraph models is *polynomial*, and the size of the result is *linear* in the size of the original encoding (measured by the size of R and arity of the PNEs and predicates); this is not necessarily true in the reverse direction.

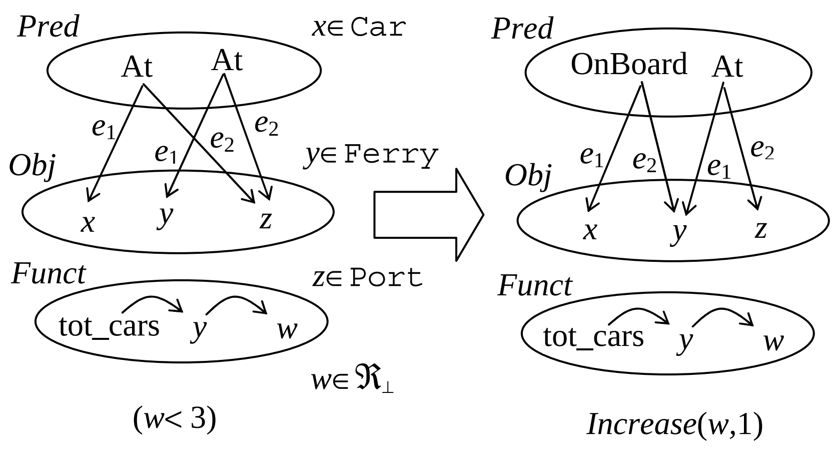


Figure 10. Theorem 1: setGraph equivalent of sentential *Board*(*x*,*y*,*z*) operator (Ferry domain)

The Hybrid Planning Representation

The hybrid representation combines, orthogonally and in a straightforward way, the analogical and sentential models described in the previous sections. In the hybrid representation, the world state is composed of two distinct parts: an analogical state and a sentential state. The two components are effectively two independent "sub-states", much like logical and numerical states are in the sentential (PDDL2.1) model. The hybrid model essentially "glues" together a setGraph state with a propositional state (containing also a vector of numeric values) and treat them as separate entities for reasoning about possible state transformation. Hence, hybrid operators (preconditions and effects) will consist of two distinct parts, each describing a transformation of the respective sub-state. Notice that any of these parts may be empty (for example, an operator could have purely analogical preconditions and purely sentential effects). The issue of how to guarantee that the state changes specified by each sub-part are *sound* with respect to the actions that they represent is dealt with in the next section. In this section, we illustrate with an example how hybrid planning works, and discuss the advantages of using a hybrid representation as opposed to a purely sentential or purely analogical one.

Example 5.3 – Consider an extended Ferry domain (see Examples 5.1 and 5.2) containing several ports, some of which are situated in proximity of petrol stations and restaurants. The ferry (which can carry a limited number of cars) must take each car to a specific port. Some of the cars, however, may need to refuel or stop for food. Cars can be taken directly to their destination if such port provides the service(s) they need; otherwise, they must first get to a port that has a petrol station and/or a restaurant, and then be taken to their destination. The possible actions of the domain are *sail*, *board* and *debark* (seen before) plus the two actions *refuel* and *eat*, consisting of filling up the car's tank and dining, respectively.

This domain could be entirely represented using the purely analogical or purely sentential models. We choose to encode the "transportation" aspects (first three actions) using setGraphs, and the "stationary" state changes (last two actions) using a sentential description; as discussed below, this choice is expected to lead to speed ups in performance.

Figure 11.(*a*) depicts the *analogical* part I_A of a possible initial state for a Ferry problem with four cars and four ports (the domain could be easily augmented with multiple ferries). Figure 11.(*b*) contains the type hierarchy for nodes (the *PLACE* part is essentially identical to that of Figure 7.(*b*)).

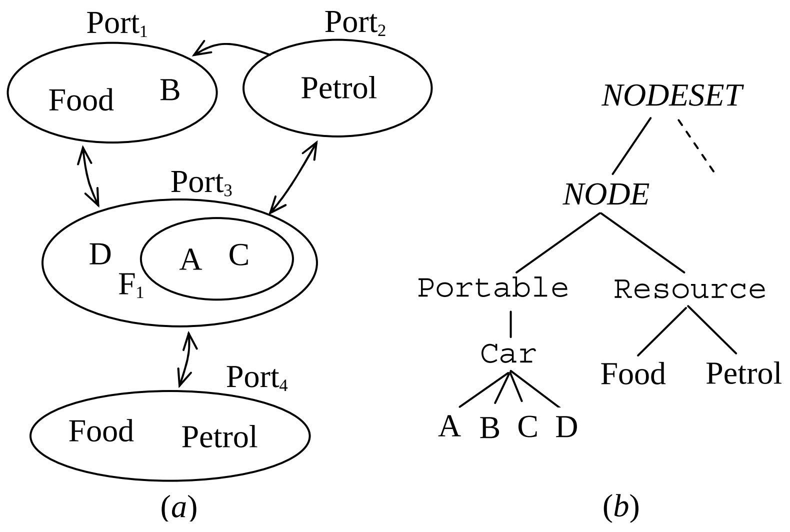


Figure 11. (*a*) Analogical initial state I_A for the Ferry domain;
(*b*) associated *NODE*-type hierarchy (see also Figure 7.(*b*))

The *sentential* part I_S of the state consists of the following set:

$$I_S = \{ \texttt{Needs}(\text{A, Food}), \texttt{Needs}(\text{A, Petrol}), \texttt{Needs}(\text{C, Petrol}) \}$$

In the initial state, car A needs both petrol and food, while car C only needs to refuel. In order to represent the number of cars currently on board of a ferry, we use a functor symbol of one argument, ‘`tot_cars`’. Accordingly, the cell of the numeric vector of the (sentential) state corresponding to the PNE `tot_cars`(F_1) is initialised to the value 2.

The *goal* of a problem will require that each car is transported to a specific location (port), and that none of the cars is left in need of any of the resources. While the former part of the goal will be described *analogically* using a setGraph, the latter is specified *sententially* by the set G_S of propositions G_S=$\{\neg\,\texttt{Needs}(x, \text{Food}), \neg\,\texttt{Needs}(x, \text{Petrol}) \mid x \in \texttt{Car}\}$.[21]

Let us now consider the set of operators. The representation of the actions *sail* and *board* (or *debark*) is essentially identical to the analogical operators illustrated in Example 5.1 and 5.2 (except that the precondition restricting the applicability of *Board* is expressed here as `<(tot_cars(`F_1`), 3)`; similarly for the numeric effect which increases such value by 1). The actions *refuel* and *eat* are more interesting: they can be encoded as a single operator *Get*, containing hybrid preconditions and only sentential effects:

Get(x, y, z) -- *Get resource x for car y from the current port z*
Parameters: x – `Resource;` y – `Car;` z – `Port`
P_A: $\langle\{ z\{x, y\}\}, \{ \}\rangle$
P_S: $\{$ `Needs`(y, x) $\}$
E_S: $\{ \neg$`Needs`(y, x) $\}$

The analogical part of the preconditions P_A requires that car x and resource y be located at the same port z; the sentential part P_S requires that car x be in need of resource y; the (purely) sentential effects E_S remove the literal “`Needs`(y, x) ” from the (sentential) state.

The main advantage of a hybrid planning system with respect to a purely sentential or analogical one is that it allows the domain engineer the flexibility to encode each aspect of the world using the most *efficient* representation for that aspect. For example, as shown by the experimental results, the adoption of an analogical model to describe a *move* domain can lead to significant efficiency gains, particularly when it allows decomposing the domain into smaller parts within which the search and update processes are simpler. This clearly applies to the above example: the spatial structure of the problem is decomposed into four parts, and the conditions for the local applicability and the execution of the boarding, debarking and "get" operators require only a linear number of steps (in the number of entities).

The ability of the domain modeller to use two different representation paradigms within a single system allows a second type of decomposition, based on the possibility of a hybrid operator to contain purely analogical (or purely sentential) preconditions and/or effects. In fact, suppose that the set of operators contains only two possible types of operators, namely, purely sentential and purely analogical. The goal and the initial state are also composed of two parts, analogical and sentential. When trying to achieve an *analogical* sub-goal, the algorithm can completely ignore all purely sentential operators, as they could not possibly achieve the sub-goal considered (and vice versa). Hence, if the set of operators is divided into two completely *independent* sets, the problem can be decomposed into two parts that can be solved independently and then integrated into a single plan solution.

If the set of operators cannot be divided into two completely independent subsets, the process of integration of the sub-solutions is not straightforward, and may lead to non-optimal plans. In the example above, the set of operators can be split into two almost independent parts, one containing purely analogical operators (*Board*, *Debark* and *Sail*), the other containing only one operator (*Get*) with hybrid preconditions and purely sentential effects. These two sets are not completely independent: since *Get* contains hybrid preconditions, a solution found for the sentential part of the problem could be "clobbered" by some of the effects of the analogical plan solution. In this specific case, one way to avoid this could be to force the purely sentential plan to be identified first, and then to take the state resulting from its execution as the new initial state. The resulting problem would be purely analogical, and its solution could be simply "appended" to the solution of the sentential part. However, this simple method would guarantee correctness, but not optimality.[22]

In addition, the above type of decomposition also allows the use of special-purpose methods for the efficient solution of the purely analogical, graph-navigation aspects of the problem, and the use of different search methods for the purely sentential part. This can lead to further planning performance speed-ups (see also (Fox & Long, 2001)).

In summary, with respect to purely sentential or purely analogical systems, the ability of hybrid models to encode different aspects of the world using different representations enables the domain modeller to choose the simpler and more efficient description for each aspect; moreover, hybrid descriptions may allow the *automatic* decomposition of the problem into two sub-problems, with consequent pruning of the search space. This also makes possible the application of more efficient, dedicated methods for the solution of the two sub-problems.

Soundness of Hybrid Planning

The simple juxtaposition of sentential and analogical representations, although apparently effective, does not guarantee the soundness of the model with respect to the real domain represented (Lifschitz, 1990). This section addresses this problem. In particular, it describes a theoretical framework (Definitions 4–7) in which both sentential and analogical models can be formalised, and identifies the necessary conditions (Definition 9) for any model within it to be sound with respect to the represented world. In particular, the Soundness Theorem presented at the end of this section extends to analogical and hybrid representation the theory of sound action description (Lifschitz, 1990), currently limited to purely sentential models.

Notice that the contents of this section bear no relation to the actual implementation of the hybrid model; all the constructs introduced are used purely for the theoretical analysis.

We begin with the formalisation of a language for describing the world. Following Lifschitz (1990), the world is taken to be, at any instant of time, in a certain *state*. A state is identified by a finite set I of *entities* and finite sets of relations among (and properties of) entities. A *domain* constitutes the set S of possible states in which the world can be. In order to describe a domain, we adopt a formal language $\mathcal{L} = \langle P, F, C\rangle$, where P, F and C are finite sets of relation, function and constant symbols, respectively. Each relation and function symbol of P and F can be either *numeric* or *logical*, depending on the nature of its arguments. Each non-constant symbol of $\mathcal{L}$ has a specific arity n, for some integer $n \geq 0$, which depends on the symbol. A language $\mathcal{L}$ can be associated to a *type hierarchy* that organises all symbols of C into subsets $T_1,\ldots,T_k$ such that $(\cup_{i\in\{1\ldots k\}}T_i) = C$. The *wff* of such a many-sorted language, atomic logical and numeric formulæ, are built as follows:

- *c is a term iff $c\in C$*
- *$f(c_1,\ldots, c_m)$ is a* primitive numeric expression (PNE) *iff $f\in F$ and $c_1,\ldots,c_m$ are terms*
- *$h(t_1, \ldots, t_m)$ is a* numeric expression (NE) *iff $h\in F$ and $t_1,\ldots, t_m$ are PNEs, NEs or numbers*
- *$p(c_1,\ldots, c_n)$ is a* logical atom *iff $p\in P$ and $\forall i\in\{1,\ldots n\}$, c_i is a term*
- *$q(t_1,\ldots, t_n)$ is a* numeric atom *iff $q\in P$ and $\forall i\in\{1,\ldots n\}$, t_i is a PNE or a NE*

The symbols of $\mathcal{L}$ are given an interpretation in the domain of interest (Chang & Keisler, 1977; Section 1.3). In particular, the interpretation function g will map each constant symbol $c\in$ C to a distinct entity $g(c)=i\in I$, each m-placed logical function symbol $f\in F$ to a function $g(f) = f'\colon I^{m} \rightarrow \Re$, and each n-placed logical relation symbol $p\in P$ to a relation $g(p) =p'\subseteq I^{n}$. In addition, g also maps each m-placed numeric function symbol $h\in F$ to a (fixed) function $g(h)$: $\Re^m \rightarrow \Re$, and each n-placed numeric relation symbol $q\in P$ to a (fixed) relation on real numbers g$(q) \subset \Re^n$. Notice that the value and truth of $f'(i_1,\ldots i_m)$ and $p'(i_1,\ldots i_n)$ may depend on the current state. In what follows we assume that, for a given domain and language $\mathcal{L}$, a *fixed* interpretation function g is adopted. The function g allows to determine, for each state s, which atoms of $\mathcal{L}$ are *satisfied* in this state and the value of any PNE and NE:

Definition 4 (Atom-satisfaction) *Given a language $\mathcal{L}=\langle P,F,C\rangle$ for a domain S, an atom $p(t_1,\ldots, t_n)\in \mathcal{L}$ is* satisfied *in $s\in S$ iff, in state s, $g(p) \supseteq (g(t_1),\ldots,g(t_n))$.*

Let $g(t)=t$ for any $t\in\Re$. If $t=f(t_1,\ldots, t_m)$, with $f\in F$ and $t_i\in C \cup$ PNE $\cup$ NE, then $g(t)$ is defined as the value of $g(f)$ in the current state $s\in S$ calculated in $(g(t_1),\ldots,g(t_m))$ (written $f(t_1,\ldots t_m)\,|_s$).

Consider an abstract data structure $\mathcal{D}$ (such as a tree, a list, a graph, etc.) and a universe $\mathcal{U}$ of elements (e.g., characters, booleans, integers, and so forth). Let $\mathcal{D}_{\mathcal{U}}$ be a select set of instances of $\mathcal{D}$ possibly containing elements of $\mathcal{U}$ (e.g., trees of booleans, lists of integers, etc.). Let $\Re_\perp = \Re \cup\{\perp\}$, where $\Re$ is the set of real numbers. The elements of $\Re_\perp$ will be called *R-values*.

Definition 5 (Model) *Given a language $\mathcal{L}=\langle P,F,C\rangle$ and a set $\mathcal{D}_{\mathcal{U}}$ of data structure instances with elements in $\mathcal{U}$, a* model *is a pair $M=(d,\varepsilon)$ where $d\in \mathcal{D}_{\mathcal{U}}$ and $\varepsilon{:}C\rightarrow \mathcal{U}$ is a 1-1 total function mapping symbols of C to elements of the universe $\mathcal{U}$.*

A model is essentially a data structure containing elements taken from a set $\mathcal{U}$. The function ε maps the relevant objects (symbols) of the domain to the corresponding elements of the universe that represent them (which may or may not appear in the model). The use of an unspecified data structure $\mathcal{D}$ allows this definition to be used for both sentential and analogical (setGraph) models, as demonstrated in Examples 5.4 and 5.5 below.

Definition 6 (Domain representation structure) *A* domain representation structure (DRS) *for a language $\mathcal{L}=\langle P,F,C\rangle$ is a triple $\langle \mathcal{D}_{\mathcal{U}},\Psi, \Phi\rangle$, where $\mathcal{D}_{\mathcal{U}}$ is a set of instances of a data structure $\mathcal{D}$ with elements in $\mathcal{U}$ and each $\psi_i\in\Psi$, $\phi_j\in\Phi$ are algorithms associated to the relation and function symbols $i\in P$, $j\in F$, respectively, such that ψ_i ,ϕ_j always terminate, and:*

- *for each n-placed logical relation symbol $p\in P$, $\psi_p : \mathcal{D}_{\mathcal{U}} \times \mathcal{U}^n \rightarrow \{True, False\}$*
- *for each m-placed logical function symbol $f\in F$, $\phi_f : \mathcal{D}_{\mathcal{U}} \times \mathcal{U}^m \rightarrow \Re_\perp$*
- *for each n-placed numeric relation symbol $q\in P$, ψ_q is such that $\psi_q:(\Re_\perp)^n\rightarrow\{True,False\}$, and $\psi_q(x_1,\ldots,x_n)= g(q)(x_1,\ldots, x_n)$ if $x_i \neq\perp$ for all $i\in \{1,\ldots n\}$, $\perp$ otherwise*
- *for each m-placed numeric function symbol $p\in F$, $\phi_h:(\Re_\perp)^n\rightarrow\Re_\perp$, and $\phi_h(x_1 ,\ldots,x_m)=\perp$ if $g(h)(x_1,\ldots,x_m)$ is undefined or if there exists x_i such that $x_i =\perp$; $g(h)(x_1,\ldots,x_m)$ otherwise*

Basically, a DRS consists of a data structure and a set of algorithms for checking it. Each algorithm takes as input a model (a data structure instance) and a set of object symbols, and (always) returns a value. For example, given *n* objects $c_1,\ldots,c_n$, in order to establish whether $p(c_1,\ldots,c_n)$ holds in the current model *M*, it will be sufficient to apply the corresponding procedure ψ_p to *M*, using symbols $\varepsilon(c_1),\ldots,\varepsilon(c_n)\in\mathcal{U}$ (representing $c_1,\ldots,c_n$ in *M*) as input.

Notice that procedures ψ_q and ϕ_h associated to the *numeric* (function and relation) symbols calculate the same truth (or numeric) value of the corresponding relations and functions, which are fixed for the chosen domain and do not depend on the current state.

Definition 7 (Model representation) *Given a language $\mathcal{L}$, a DRS $\mathcal{R} = \langle\mathcal{D}_{\mathcal{U}},\Psi,\Phi\rangle$ for $\mathcal{L}$ and a model $M=(d,\varepsilon)$ in $\mathcal{R}$ (i.e., such that $d\in\mathcal{D}_{\mathcal{U}}$),* M represents *a state $s\in S$ (written $M \cong_{\mathcal{R}} s$) iff, for every logical atom $p(t_1,\ldots,t_n)$ and PNE $f(t_1,\ldots, t_m)$ of $\mathcal{L}$, both of the following conditions hold:*

- *$\psi_p(d, \varepsilon(t_1),\ldots, \varepsilon(t_n))$ =True iff $p(t_1,\ldots, t_n)$ is satisfied in s*
- *$\phi_f(d, \varepsilon(t_1),\ldots, \varepsilon(t_m)) =f(t_1,\ldots t_m)|_s$ if $f(t_1,\ldots t_m)$ is defined in s, $\perp$ otherwise*

Example 5.4 – Consider a BW domain in which blocks have a specific *weight*; the blocks and a table are the entities of interest, 'to be on' is the relevant relation between entities, and the weight of a block is the only property of interest. The language

$$\mathcal{L}_1= \langle P_1,F_1,C_1\rangle = \langle\{On, \geq\}, \{Weight, +, -, * , /\}, \{\mathrm{T}, \mathrm{B}_1, \mathrm{B}_2, \mathrm{B}_3\}\rangle$$

with types *Block*=$\{\mathrm{B}_1, \mathrm{B}_2, \mathrm{B}_3\}$ and *Table*=$\{\mathrm{T}\}$ can be adopted to reason about a BW domain with three (weighted) blocks. *Weight* is a 1-placed logical function symbol with argument in *Block, On* is a 2-placed logical relation symbol with unrestricted argument type. '≥' is a 2-placed numeric relation symbol, and +, –, * and / are 2-placed numeric function symbols. The interpretation *g* of the symbols of $\mathcal{L}_1$ is intuitive: *Weight* denotes the function *Block* $\rightarrow\Re$ returning the weight of a block, ≥ is the binary relation *greater than or equal to* defined on $\Re$, and +, –, *, / are the standard arithmetic operations on $\Re$. *On* is mapped to the corresponding spatial relation between objects (blocks and table).

Let us build, for this domain and language, a *sentential* domain representation structure DRS$_1$ which replicates the semantic model of PDDL2.1-*lev2**. Accordingly, we represent the state using a data structure $\mathcal{D}_1=(L,R)$ composed of a set L of logical atoms of $\mathcal{L}_1$ (built using the terms of $\mathcal{L}_1$) and a vector R of three cells (with values in $\Re_\perp$). Hence, $\mathcal{U}_1=C_1\cup\Re_\perp$.

Procedure $\psi_{On}(d,x,y)$ takes as input $d=(l,r)$, an instance of $\mathcal{D}_1$, and two elements $x,y\in C_1\subset \mathcal{U}_1$ and returns *True* iff $On(x,y)\in l$. Procedure $\psi_{\geq}(x,y)$ takes as input two R-values and returns *True* if x is equal to or greater than y, *False* if x is smaller than y, $\perp$ otherwise. Procedure $\phi_{Weight}(d,x)$ takes as input $d=(l,r)$, an instance of $\mathcal{D}_1$, and an element $x\in C_1\subset \mathcal{U}_1$ and returns the value of $r[0]$ if $c=\mathrm{B}_1$, $r[1]$ if $c=\mathrm{B}_2$, $r[2]$ if $c=\mathrm{B}_3$, $\perp$ otherwise. The procedures ϕ_+, ϕ_-, ϕ_* and $\phi_/$ take two R-values and return the result of the corresponding operation applied to the input if such result is a real number, $\perp$ otherwise.

Then, given a model $M=(d,\varepsilon)=((l,r), \varepsilon)$ such that $\varepsilon{:}C_1\rightarrow \mathcal{U}_1$ is defined as $\varepsilon(x)=x$ for all $x\in C_1$, M represents a state s of the domain iff l contains all and only the logical atoms of $\mathcal{L}_1$ which are satisfied in s, and cells $r[0]$, $r[1]$, $r[2]$ of vector r contain the values corresponding to the weights of the three blocks of the domain. This encoding is analogous to the semantics of the corresponding PDDL2.1 representation of this domain (Fox & Long, 2003).

Notice that in a certain state s one or more of the entities of interest might not exist at all. For example, in BW one of the actions could have the effect of destroying (or "consuming") a block (resource). A model of a BW state in which the i-th block does not exist should have $r(i-1)$ set to $\perp$, so that $Weight(x)$ is evaluated $\perp$ if the block identified by x does not exist.

If M represents state s, it should be possible to use procedure $\psi_{\geq}$ to determine whether any arbitrarily-complex numeric atom of $\mathcal{L}_1$ is satisfied in s – e.g., whether $\geq(\ *(Weight(\mathrm{B}_2),2.5), Weight(\mathrm{B}_1))$ is satisfied. However, Definition 7 only requires that the procedures calculating the PNEs (here, ϕ_{Weight}) and the logical atoms return the "correct" value. Nevertheless, this is sufficient to guarantee that also all NEs and all possible numeric atoms of $\mathcal{L}_1$ are calculated correctly, as the last two points of Definition 6 require that procedures ψ_q (here, $\psi_{\geq}$) and ϕ_h (here, ϕ_+, ϕ_-, ϕ_* and $\phi_/$) return the value of the corresponding relations and functions on $\Re$.

An *action* is a function $a{:}S\rightarrow S$ that transforms each state $s\in S$ into a state $s'= a(s)\in S$. Although there might be some $s\in S$ such that $a(s) = s$, we assume that $a(s)$ is always defined.

Definition 8 (Planning domain) *A* planning domain *is a pair $\langle S,A\rangle$, where S is the set of possible states in which the world can be, and A, the set of actions, is a finite set of total functions $a{:}S\rightarrow S$.*

Given a planning domain $\langle S,A\rangle$ (with language $\mathcal{L}$ and DRS $\mathcal{R}$), a set of models Σ (in $\mathcal{R}$) is said to *represent* the set of states S (written $\Sigma \cong_{\mathcal{R}} S$) *iff* for each model $M\in\Sigma$ there is one (and only one) state $s\in S$ such that $M \cong_{\mathcal{R}} s$, and for each $s\in S$ there is one model M such that $M \cong_{\mathcal{R}} s$.

Given a set of models Σ representing the set of states S, an action $a{:}S\rightarrow S$ can be modelled as a function $\alpha{:}\Sigma\rightarrow\Sigma$ transforming (corresponding) model M into (corresponding) model M':

Definition 9 (Sound action model) *Given a domain $\langle S,A\rangle$ (with language $\mathcal{L}$ and DRS $\mathcal{R}$) and a set Σ of models in $\mathcal{R}$ such that $\Sigma \cong_{\mathcal{R}} S$, a function $\lambda{:}\Sigma\rightarrow\Sigma$ is* sound *with respect to action $a{:}S\rightarrow S$ iff, for each model $M\in\Sigma$ and state $s\in S$ such that $M \cong_{\mathcal{R}} s$, $\lambda(M) \cong_{\mathcal{R}} a(s)$.*

For a function λ to be sound w.r.t. action a, it must map each model M (representing state s) into the model M' that represents the state obtained from the application of action a to s.[23]

Given a domain $D=\langle S,A\rangle$, a pair $R=\langle\Sigma,\Lambda\rangle$ is a *sound representation* of D *iff* Σ is a set of models representing S, and $\Lambda=\{\lambda_1,\ldots \lambda_k\}$ is a set of sound models of the actions $\{a_1,\ldots a_k\}=A$.

Theorem 2 (Soundness) *Let $R=\langle\Sigma,\Lambda\rangle$ be a sound representation of a domain $D=\langle S,A\rangle$. Let $\lambda=\langle\lambda_1,\ldots\lambda_n\rangle$ (with $\lambda_i\in\Lambda$) be a sequence (plan) of sound action models, and $a=\langle a_1,\ldots a_n\rangle$ (with $a_i\in A$) be the corresponding sequence of actions. If $M_0\in\Sigma$ represents $s_0\in S$, and the application of λ to M_0 produces $M_n=\lambda(M_0)=\lambda_n\circ\ldots\circ\lambda_1(M_0)$, then M_n represents $a_n\circ\ldots\circ a_1(s_0)$.*

Proof – The proof is by induction, and it is analogous to the original version (Lifschitz, 1990) except that the concept of *satisfaction*, limited to sentential models, is replaced here with that of *model representation* (Definition 7), applicable to both sentential and analogical models.

The basic case (n=1) follows immediately from Definition 9. Assume that the theorem holds for $n=k$, and let us see that it holds for $n=(k+1)$. Let $M_0\in\Sigma$ *represent* $s_0\in S$. If $n=(k+1)$, then $M_n=M_{k+1}=\lambda_{k+1}\circ\lambda_k\circ\ldots\circ\lambda_1(M_0)$. Because of the inductive hypothesis, $\lambda_k\circ\ldots\circ\lambda_1(M_0)=M_k$ represents state $s_k=a_k\circ\ldots\circ a_1(s_0)$. Since λ_{k+1} is sound with respect to a_{k+1}, by definition of sound action, $\lambda_{k+1}(M_k)\cong_R a_{k+1}(s_k)$. In other words, $\lambda_{k+1}\circ\ldots\circ\lambda_1(M_0)$ represents $a_{k+1}\circ\ldots\circ a_1(s_0)$. Q.E.D.

Example 5.5 – Consider the BW domain of Example 5.4, with the same language $\mathcal{L}_1$ and interpretation specified there. Let us define, for this domain and language, an *analogical* domain representation structure DRS$_2$. The data structure $\mathcal{D}_2$ adopted to describe the world state is the setGraph. In particular, Figure 12.(*a*) depicts the encoding used to describe a BW state with three blocks, having weight 2.5, 0.6 and "unknown" ($\perp$). The universe $\mathcal{U}_2$ of (node) symbols is identical to $\mathcal{U}_1$ (*viz.*, $\mathcal{U}_2=C_1\cup\Re_\perp$). The associated *NODE* type-hierarchy is depicted in Figure 12.(*b*) (the *PLACE* part contains only instances P$_1$,…P$_{10}$).

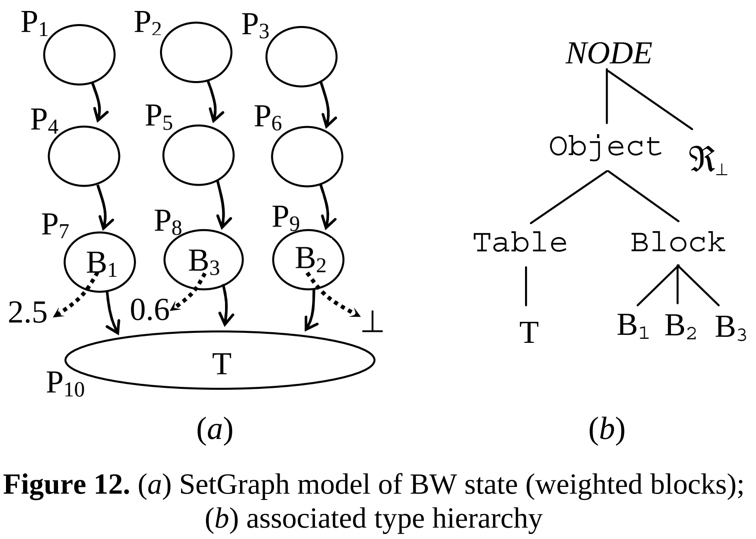


Figure 12. (*a*) SetGraph model of BW state (weighted blocks); (*b*) associated type hierarchy

The procedures $\psi_{On}(d,x,y)$ and $\phi_{Weight}(d,x)$ are encoded using the *parameterised* setGraphs G_{On} and G_{Weight} depicted in Figures 13.(*a*) and (b), respectively. In particular, procedure $\psi_{On}(d,x,y)$ takes as input an instance d of $\mathcal{D}_2$ (ground setGraph), and two symbols $x,y\in\mathcal{U}_2$ and returns *True* iff the setGraph $G_{On}(x,y)$ (having the parameters replaced by the corresponding input symbols) is *satisfied* in d. Procedure $\phi_{Weight}(d,x)$ takes as input an instance d of $\mathcal{D}_2$ and a symbol $c\in\mathcal{U}_2$ and, if there is a function σ and a variable substitution θ such that setGraph $G_{Weight}(x,w)$ is satisfied in d with mapping σ and substitution $\theta=(x/c,\ w/val(\sigma(w)))$, it returns the value of $\sigma(w)$. Procedures $\psi_\geq$, ϕ_+, ϕ_-, ϕ_* and $\phi_/$ are defined as in Example 5.4.

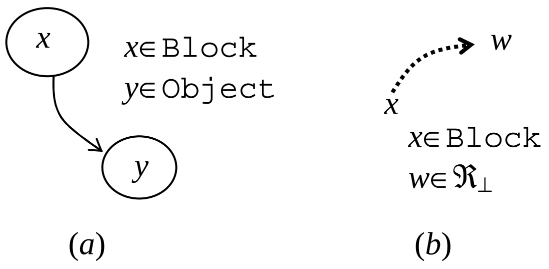


Figure 13. Parameterised setGraphs: (*a*) G_{On}; (*b*) G_{Weight}, encoding, respectively, procedures ψ_{On} and ϕ_{Weight}

Let Σ_1 be the set of models (G,ε), where $\varepsilon:C_1\rightarrow \mathcal{U}_2$ is such that $\varepsilon(x)=x$ for all $x\in C_1$, and G can be any of the ground setGraphs obtainable from the one specified in Figure 12.(*a*) by moving nodes B_1, B_2 and B_3 from their places to any other of $P_1,\ldots,P_9$ (allowing at most one node in one place, and no pair of places u,v connected by an $On(u,v)$ edge such that u contains a node and v does not). Let Λ_1 be the set of functions $\lambda_{x,y,z}:\Sigma_1\rightarrow\Sigma_1$ defined by the result of the application of the analogical operator $Move(x,y,z)$ (Figure 2.(*b*)) to the models of Σ_1, for each possible $x\in$ `Block`, $y,z\in$ `Object` (if $Move(x,y,z)$ is not applicable, we define $\lambda_{x,y,z}(x)=x$).

Let A_1 be the set of possible actions $Move_{x,y,z}$ of the BW domain (consisting of picking up a block x from the top of an object y and putting it onto object z), and let S_1 be the set of possible BW states that can be obtained by applying them to a legal initial state (if a move is not applicable, it leaves the state unaltered). Then, the pair $R_1=\langle\Sigma_1, \Lambda_1\rangle$ is a *sound representation* of the domain BW=$\langle S_1, A_1\rangle$. In fact, because of the way in which they have been built, the models of Σ_1 represent the states of S_1. In addition, every function $\lambda_{x,y,z}\in\Lambda_1$ is *sound* with respect to the action $Move_{x,y,z}$.

Therefore, in virtue of the Soundness Theorem, the domain description $\langle\Sigma_1,\Lambda_1\rangle$ can be used (in conjunction with the DRS$_2$ defined above) to generate sound plans for the BW domain.

Given the ability of the theory to formalise both sentential (Example 5.4) and analogical (Example 5.5) models, it is easy to show that it can also formalise *hybrid* models. Hybrid models (e.g., see Example 5.3) will represent a state as a data structure $\mathfrak{D}$ containing two elements: a sentential state (composed of a set of ground atoms and a vector of R-values) and an analogical state (a ground setGraph). The domain representation structure can be defined, for the language considered, using either sententially- or analogically-based procedures (see Example 5.4 and 4.5, respectively). So can the action descriptions. We conclude that the conditions identified by Definition 9 can be considered as conditions for sound sentential, analogical and hybrid models of action; similarly, the Soundness Theorem can be applied equally well to sentential, analogical and hybrid planning representations.

RELATED WORK

The work of Glasgow and Malton (1994) on purely analogical, model-based spatial reasoning is closely related to the ideas adopted in the proposed framework. Glasgow and Malton describe a representation for spatial reasoning based on array theory (More, 1981) in which symbolic arrays depict the entities and relations of the world:

“An array consists of zero or more symbols held at positions along multiple axes, where rectangular arrangement is the concept of objects having spatial positions relative to one another in the collection. In order to specify spatial relations, a symbol may occupy one or more cells of an array." (Glasgow & Malton, 1994, p.7)

The authors provide a semantics for their representation by requiring that, for a world to be represented by an array, a mapping between symbols in the array and entities in the world exists that preserves the relative location of entities. In particular, they specify a set Ψ of fixed, "primitive" boolean array functions for inspecting an array, where each function is associated to a spatial relation of interest in the world. An n-ary spatial relation r_i is said to be *represented* in an array A by the corresponding function ψ_i when $\psi_i(s_1,\ldots, s_n)$ returns *True* iff $(s_1,\ldots, s_n)\in r_i$ (where $s_1,\ldots,s_n$ are symbols denoting entities). An array representation is a *model* for a world if each relation r_i is represented by the corresponding array function $\psi_{i.}$ This idea is clearly at the basis of the concept of model representation adopted here (Definition 7). However, in addition to being used only for purely analogical models, the set of primitive transformation functions proposed by Glasgow and Malton for manipulating arrays is *fixed* and predetermined. By providing a formalism that allows the domain modeller to specify inspection and transformation procedures (e.g., see Figure 13), the present work generalises and extends that of Glasgow and Malton's.

Myers and Konolige (1995) present a hybrid framework for problem solving that allowed a sentential system (using a first-order logic language) to carry out deductive reasoning with and about diagrams. In their framework, any analogical representation S is described by a set of first-order *diagram models*, constituting all the possible completions of the partial information provided by S. A diagram model consists of a set of binary analogical relations $A\subseteq E_s \times E_s$ and a set of label relations $L \subseteq E_s \times E_l$, with E_s the set of diagram elements and E_l the set of labels. The analogical relations encode the "structure" of the diagram (the spatial relations between the elements), while the label relations are used, for example, to assign a type (or any other label) to elements of the diagram. Myers and Konolige provide a theoretical analysis of the properties (soundness, equivalence and completeness) of their framework, assuming that, for a given analogical structure, sound and complete *reflection* and *extraction* procedures are given, which allow, respectively, the monotonic addition of information to and extraction of information from diagram models. The extraction procedures are essentially equivalent to the inspection procedures Ψ used in setGraphs. Reflection procedures, on the contrary, do not have a direct equivalent in setGraphs; they allow transforming a set of diagram models containing structural *uncertainty* into one that is (strictly) more determined by updating it with information obtained from the sentential deductive process. Most importantly, however, Myers and Konolige's model does not permit existing analogical information to be "retracted" from the diagram models. This possibility is crucial for enabling *nonmonotonic* changes of a diagram, typically associated with the execution of an action, and, hence, required by a system that must be able to plan. Similar considerations also apply to works on heterogeneous (hybrid) representations, such as (Barwise & Etchemendy, 1998; Swoboda & Allwein, 2002).

The work of Forbus (1995) and colleagues (Forbus *et al*., 1987, 1991) on qualitative spatial reasoning is also relevant in this context. Forbus proposes a Metric Diagram/Place Vocabulary (MD/PV) model of reasoning, in which a "purely qualitative" representation (PV) is extracted from an underlying metric diagram, containing all the necessary numerical information required for the task at hand. The PV is then used to support abstract, qualitative reasoning about motion, while the MD provides the information required to calculate more precise conditions for detailed predictions. There is a clear similarity between PV and places in setGraphs. According to Forbus' definition of "not purely qualitative" ("… representations whose parts contain enough detailed information to permit calculation […]" (Forbus, 1995, p.185)), setGraphs are not purely qualitative, but rather hybrid domain descriptions, in which the numeric elements (representing some of the metric information "extracted" from the MD) are integrated in the qualitative model. With respect to the MD/PV model, setGraphs offer the advantage of a single, unified formalism of representation, in which qualitative and quantitative information are integrated to support both types of reasoning, without requiring the use of an underlying metric diagram.

SetGraphs are closely related to semantic network representations (Lehmann, 1992). For example, Sowa's Conceptual Graphs (Sowa, 1984), a formalism expressively equivalent to first-order logic, can be easily encoded using setGraphs. Petri nets (Petri, 1963) can also be naturally represented using a setGraphs. In fact, assume that the *tokens* of a Petri net are described as setGraph nodes. Petri-net places ("passive nodes") can be encoded by setGraph places, while *transitions* ("active nodes") can be represented as a specific type of node (let us call it *Trans*). Figure 14 shows a setGraph operator encoding the movement of a *single* token in any Petri net. The simulation of the *parallel* movement of several tokens can be represented using similar action schemata.

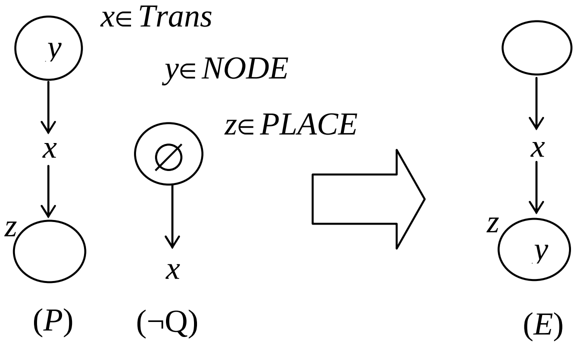


Figure 14. Petri net dynamics encoded by a setGraph action schema. Precondition (¬Q) requires all inputs to *x* to contain at least one token

Notice that in order to represent Petri net dynamics, the setGraph formalism needs to be extended with a symbol ("∅") that allows explicitly requiring a place to be empty, and by introducing *negative* preconditions (all setGraphs in the negative preconditions must be *not satisfiable* in a state *s* for the operator to be applicable in *s*).

Another example of diagrammatic structure similar to the setGraph is the "higraph" (Harel, 1988), based on a combination of Euler/Venn diagrams and generalised graphs. Higraphs can represent subset relations, Cartesian product relations and arbitrary relational assertions (through labelled arcs), and are amenable to a wide variety of uses. While most features of higraphs can be replicated in a setGraph by making use of the type hierarchies, the possibility for a place (roughly equivalent to the concept of "blob" in a higraph) to overlap only *partially* with another place is not envisaged in the proposed definition of setGraph. However, it should not be too difficult to extend the definition to allow also this feature.

In the area of planning, the proposed approach has close links with the work of Long and Fox (2000) on generic types and on their use in problem decomposition (Fox & Long, 2001). Long and Fox have developed domain analysis techniques that allow the automatic identification of the different, generic types of objects (e.g., "mobiles", "portables") of a planning domain from its purely sentential description. These techniques dovetail nicely with the present framework. In fact, once the different generic types of objects of a domain have been isolated, hybrid planning models can be used to encode and solve them using different representations for different generic types. The work of Long and Fox has also demonstrated that many domains are *isomorphic* to and can be treated as "transportation" or "construction" problems even when this is not apparent from their original description. If the dynamics a domain can be automatically recast in terms of movement or manipulation of (possibly abstract) objects, hybrid or analogical representations can be adopted to solve them efficiently (possibly by adopting special-purpose, graph-traversal algorithms). For example, if *activities* are represented as mobile objects, and locations denote s*ynchronisation points* or *intervals*, then the problem of scheduling a number of tasks over a given time period can be recast as that of assigning to each "object" (activity) an appropriate "location" (start/end time point), subject to various numerical constraints. (This idea was illustrated earlier by the use of setGraphs for encoding Petri nets – see Figure 14).

In the attempt to address the inefficiencies caused by the ramification problem that plague sentential planning languages based on classical logic (i.e., STRIPS (Fikes & Nillsson, 1971) and ADL (Pednault, 1989)) several researchers (e.g., (Lifschitz, 2002; Dimopoulos, Nebel & Koehler, 1997; Erdem & Lifschitz, 1999; Subrahmanian & Zaniolo, 1995; Gelfond & Lifschitz, 1993) have investigated the possibility of reducing the planning problem to the problem of finding an answer set ("stable model") for a logic program. The alleged advantage of this approach is that the representation of properties of actions is easier than in STRIPS or ADL, in view of the fact that, in logic programs, domain axioms are no different from any other of the rules of the program. However, although this removes the need to describe the indirect effects of an action in the effects, it still requires the system to include such axioms explicitly in the description and take into account during the reasoning process (see the first section of this chapter). In addition, the adoption of logic programs appears to be a step backwards in the solution of the frame (and ramification) problem. In fact, not only must trivial axioms (encoding rules such as "a block cannot be in two places at the same time") still be added to the description as explicit, "pointable" formulæ: now, *frame* axioms (e.g., "a block which is not moved remains where it is") must also be included (see the "inertia" rule of Figure 3 in (Lifschitz, 2002, p.50)). In contrast, analogical (or hybrid) representations allow such axioms to become implicit constraints of the representation and actually *disappear* from the description (see the fourth section of this chapter). Hence, the logic programming approach still raises serious concerns in terms of scalability. So does the use of SAT-based approaches (briefly introduced in the first section) in conjunction with "high-level" action languages (Giunchiglia & Lifschitz, 1998) and "tight" logic programs (Erdem & Lifschitz, 2003) to perform answer set programming without answer set solvers.

DISCUSSION

Research in AI and knowledge representation has long since demonstrated that the type of formalism adopted plays a fundamental role in determining the difficulty of reasoning and problem solving (e.g., (Amarel, 1968; Simon, 1981; Larkin & Simon, 1987). Several authors have advocated the advantages of diagrammatic representations with respect to sentential ones (e.g., (Koedinger, 1992; Kulpa, 1994; Glasgow *et al*., 1995)) and the flexibility of heterogeneous models with respect to each of these formalisms alone (e.g., (Barwise & Etchemendy, 1995, 1998; Swoboda & Allwein, 2002)). However, the domain modelling languages developed for action planning have remained, throughout history, purely sentential (Fikes & Nillsson, 1971; Pednault, 1989; Fox & Long, 2003).

The main contributions of this chapter are a practical proposal and an underlying theoretical framework for sound, hybrid planning. The model for integration of sentential and analogical representations consists of the simple juxtaposition of the two formalisms in state, action and goal descriptions. The conditions for the soundness of such hybrid models (Definition 9) are based on the concept of *model representation*, which is relatively simple to use in practice (see Example 5.5). Importantly, these conditions – and, indeed, the entire theory described in the “Soundness of Hybrid Planning” section – are *not specific* to the sentential or analogical models that have been considered. Although we have shown how both setGraphs and PDDL2.1-*lev2** formalisms can be represented within this framework, the hybrid model proposed provides a basis for the integration of *any* sentential and diagrammatic description that fit its premises. For example, it should be relatively straightforward to extend the two representations to more expressive formalisms by introducing additional features such as quantification, conditional effects and disjunctive and negative preconditions (an example of the latter was presented in the last section, Figure 14). The two resulting formalisms would still be able to be integrated using the hybrid model proposed, even if they were not expressively equivalent.[24]

The second main contribution of this chapter is an analogical planning representation (Definitions 1–3) based on setGraphs, and a theoretical result demonstrating the equivalence of this formalism to a propositional planning language with functions, variables and numeric values (Theorem 1).[25] Although examples of analogical and hybrid operators encoded textually were given, respectively, in Examples 5.2 and 5.3, a detailed, specific syntax for setGraph or hybrid planning languages was not discussed here. A BNF specification of a syntax for a purely analogical planning description language is proposed in (Garagnani and Ding, 2003), but is restricted to an array-based representation analogous to the one adopted by the ABP planner. While the full setGraph representation certainly requires a more complex definition, the simplicity of the elements upon which the model is built – namely, sets and graphs – should make a syntax specification relatively straightforward.

SetGraphs are simple but expressive data structures that have the ability to *implicitly* encode the basic properties and constraints of physical domains and to reflect their inherent (topological or semantic) *structure*. Because of these features, they can lead to more efficient problem encodings, particularly when a domain can be decomposed into smaller parts that enable a “localised” search and state update operations. In addition, as discussed in the section “Advantages and Limitations of Analogical Models”, setGraph (and, in general, analogical) representations help ease the *ramification* problem by implicitly embodying constraints that sentential representations must make explicit.

An important issue concerning knowledge representation languages for common-sense reasoning is that of *elaboration tolerance*. According to John McCarthy[26], a “formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena”. There are different degrees of elaboration tolerance. For example, the Ferry domain description of Example 5.3 added new constraints to the description given initially in Example 5.1 (namely, by saying that ports can have petrol stations and restaurants, and that cars can stop at their destination only if they have acquired such resources). Some formalisations would require complete rewriting in order to accommodate this elaboration; others (like natural languages) have the ability to allow the elaboration by an *addition* to the previous encoding. SetGraphs present a high degree of elaboration tolerance: in fact, as demonstrated by Example 5.3, the encoding of the new version of the Ferry domain subsumes the encoding adopted for the original version; in other words, the additional requirements lead simply to the old representation to be *extended* with new entities, actions and constraints. This example is not just an isolated case: it is easy to see that the model could be conveniently modified to include multiple ferries, a limited amount of fuel, cars with attributes (e.g., color, size, weight), and so forth.

The above considerations indicate that, in addition to being often more efficient than sentential encodings, analogical representations can be as expressive and flexible as propositional languages. The integration of setGraphs into a *hybrid* model makes the planning formalism even more powerful. As discussed after Example 5.3, the main advantage of a hybrid system with respect to purely sentential or analogical ones is that it enables the domain modeller to encode each aspect of the world using the most *efficient* formalism for that specific aspect. Moreover, describing a domain using two different paradigms allows the automatic *decomposition* of the problem into two separate parts that can be solved independently and re-integrated into a single plan. Under certain conditions this process is straightforward; often, however, extra work will be required to produce a final, optimal plan.

The possibility of separating the analogical part of a domain description from the sentential one suggests that hybrid representations may also be effective in the automatic extraction of heuristics. In particular, useful heuristics can often be extracted by "relaxing" the planning instance at hand (e.g., by "ignoring", or abstracting, some of the details) and solving the simpler problem thus obtained. The solution of the relaxed problem can then be used to guide the search in the original problem space (e.g., see (Haslum & Geffner, 2000; Hoffmann & Nebel, 2001)). Ignoring the sentential component of a hybrid description yields a relaxed problem which can be solved more efficiently (possibly using graph-traversal algorithms).

Interestingly, the *learning* of heuristics and domain-specific control knowledge also appears to be facilitated by the adoption of analogical and hybrid descriptions. To see this, observe that the ability to learn from the solution of different problems in the same domain depends heavily on the capacity to recognise common "patterns" in different plan solutions. Consider the complexity of identifying such patterns in sequences of sentential state descriptions (for example, determining the existence of two identical stacks of blocks in different BW states). Analogical representations can be used to decompose the structure of the domain into simpler subparts (in BW, the stacks) that can be compared much more efficiently and effectively.

As illustrated at the end of the second section of the chapter, analogical (and, hence, hybrid) descriptions also allow move domains to be recast in ways that allow the spatial relations of the domain to become a static (or invariant) part of the domain. In addition, in virtue of their ability to contain multiple occurrence of the same object (including numeric values) and to describe actions involving non-conservative changes, setGraphs can easily represent resource production and consumption. Finally, it is worth noticing that, besides the mentioned advantages in terms of planning performance, hybrid and analogical representations also allow simpler and more "natural" descriptions, leading to planning domain encodings which are less error-prone and easier to read and modify.

The framework for hybrid planning representation proposed is still limited in many ways. For example, some of the important issues that have not been addressed here include the representation of time and durative actions, the definition of conditions for the parallel execution of multiple actions, and the ability to represent uncertainty and non-deterministic actions. In a sense, the possibility of having parameterised setGraphs introduces a form of uncertainty in the representation: a parameterised setGraph represents the set of possible ground setGraphs that can be obtained by replacing types and variables with appropriate instances in all possible ways (just like the set of diagram models of Myers and Konolige's (1995) system constitutes all the possible completions of a structurally uncertain diagram). However, the complete formalisation of a planning domain representation for hybrid models that allows uncertainty and non-determinism lies beyond the scope of this chapter. The introduction of time and non instantaneous actions in analogical models would appear to require, at first glance, action representation methods similar to those developed by Fox & Long (2003) for sentential languages. However, the introduction of time in conjunction with other features (such as continuous effects) can significantly complicate the problem for analogical models. Similarly, a precise treatment of the conditions for the parallel execution of analogical operators in the presence of any of the other issues is likely to require a more complex criterion than the one suggested in section "Advantages and Limitations of Analogical Models".

To conclude, this work represents a first step towards the introduction of hybrid and analogical representations in planning. The aim of this chapter was to provide a sound theoretical basis and a concrete proposal that could be put to use in practice to develop new, more efficient, hybrid (or analogical) domain-description languages, based on setGraphs or on other, more advanced, non-sentential structures. Although many issues still remain to be explored, it is the author's belief that further advances in the efficiency, flexibility and range of applicability of automatic planners will depend, to a large extent, on an increased co-operation and exchange of ideas between the planning and knowledge representation and reasoning communities.

ACKNOWLEDGEMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council, grant no. GR/R53432/01. The author wishes to thank Neil Smith, Yucheng Ding, David Smith, Bernhard Nebel, Jörg Hoffmann and Michael Jackson for their useful comments and discussions, the three anonymous reviewers for their recommendations on how to improve this chapter, and Asa Benstead for courageously agreeing to proof-read its final draft. Special thanks go to Maria Fox and Derek Long for their invaluable suggestions and support.

Notes

[1] A language is "inefficient" if it produces problem encodings in which the search for a solution is significantly more difficult than what it would have been if a different language had been adopted.

[2] The *Stack*(*x*,*y*) operator should be completed with a precondition requiring $x \neq y$, expressed using a predicate `Different`(*x*,*y*) (or ¬`Equal`(*x*,*y*)) whose instances should be listed in the initial state *I*, for all blocks *x*,*y*.

[3] The two main modules of a typical SAT-planner are the *compiler* and the *solver*. The compiler takes a planning problem as input, guesses a plan length and generates the propositional formula; a symbol table records the correspondence between the propositional variables and the planning instance. The solver uses systematic or stochastic methods to find a satisfying assignment, which will then be translated into a plan (using the symbol table). If the formula is unsatisfiable, the compiler generates a new encoding using a longer plan length (see (Weld, 1999) for a more in-depth description).

[4] In what follows, the terms *analogical* and *diagrammatic* are used interchangeably. The distinction between analogical and sentential representations will be clarified later on in the chapter.

[5] Notice that sub-graph G_4 could also be achieved with *Move*(B,C, z_2) by instantiating z_2 = A; however, this would then prevent P_2 from being satisfied in the initial state *I*, requiring the addition of further steps and leading to a longer plan solution.

[6] From the point of view of a practical implementation, introducing different types of edges in a graph does not represent a problem, as it is equivalent to allowing *labelled* edges. In Figure 4, the use of different styles of arcs instead of different labels denoting types avoids cluttering of the figure.

[7] According to Figure 4.(*a*), `Above`(*x*,*y*,*n*) edges hold only for $n > 1$. However, for this domain to be entirely equivalent to its sentential version, the 'above' relation must subsume the 'on' relation. This can be achieved by adding an extra `Above` edge for each `On` edge (not included to avoid cluttering).

[8] A *formal* description language does not necessarily mean sentential. For example, the notation $\{x,y,\ldots,z\}$, indicating a set containing elements *x*,*y*,…*z*, is not sentential.

[9] In particular, given a two-dimensional place A and two symbols $x,y \in U$, the expressions $A(x\uparrow y)$ and $A(x\rightarrow y)$ were used to indicate, respectively, that x,y appear in the same column and row of A; the expression $A(x/y)$ denotes two consecutive symbols on the same column. All such symbols play in this model the role that edges played in the graph-based model.

[10] See http://ipc.icaps-conference.org/

[11] An initial instantiation of the parameters of the operators in all possible ways (as performed by several modern planners adopting *planning-graph* techniques (Blum & Furst, 1997)) would produce $O(n\ m^k)$ *ground* operators, where m is the number of objects, n the number of original (parameterised) operators and k is the number of parameters. The check for applicability of a *ground* operator to a state would require $O(g\ p)$ steps, where g is the number of atoms in the preconditions and p is the number of propositions in the state. In general, p still grows as $O(m^k)$, where k is the arity of the predicates of the language; however, this can be improved by imposing an *order* on the propositions of the state, which allows, for example, binary searches. Hence, checking for the applicability of one operator instance would take only $O(g\ k \log m)$ steps. On the other hand, the polynomial number of ground operators would lead to a dramatic increase in the branching factor, offsetting these benefits.

[12] In general, the computational complexity of the procedure for verifying whether the preconditions of a setGraph operator are satisfied in a given state (setGraph) is equivalent to that of checking whether a certain graph is a sub-graph of another graph. This, in general, cannot be carried out in just a *linear* number of steps (in the number of nodes and edges). Fortunately, the topological structure of the domain can often be *decomposed* in several "linear" sub-structures (e.g., the stacks of BW). Although these substructures may be non-linearly connected, checking for the existence of specific conditions and manipulating objects *within* them only requires a linear number of steps, as illustrated by the examples. It is precisely this ability to "mimic" the topology of a domain that differentiates analogical models from sentential ones, and which allows this structuring and *decomposition* of the domain to take place.

[13] More precisely, block x is *above* y iff symbol x appears to the right of y (in the same array).

[14] In fact, for each object x with possible states $Sx = \{s_1, s_2, \ldots, s_k\}$, let the setGraph representation contain k corresponding nodes $\{n_1, n_2, \ldots, n_k\}$. The fact that object x is currently in state s_i can be represented by the presence of a symbol x in node n_i. The state-transition $s_i \rightarrow s_j$ of an object will be described by the movement of symbol x from node n_i to node n_j, obtained through the application of appropriate analogical operators.

[15] It should be underlined that the labels are just elements of the *notation* that has been adopted here for referring to nodeSet data structures and their contents. In other words, expressions (4.5) and (4.3) should be considered simply as alternative *descriptions* of the same nodeSet data structure.

[16] A numeric expression is either a real number, a numeric variable appearing in the analogical part of P, or an expression combining variables and numbers through operators +,–, *, /.

[17] Movement and removal of elements in the i-th setGraph G_i of P are encoded implicitly by the i-th setGraph G_i' of E. The different nodeSets of G_i and their (possibly new) positions are identified in G_i' using the same identifiers that those elements have in G_i. However, since addition of elements is permitted, G_i' might also contain new nodeSets (associated to labels or values that do not appear in G_i) or new edges. Similarly, since removal is permitted, G_i might contain nodeSets or edges that do not appear in G_i'.

[18] All the update operations will be calculated using the "old" values of the numeric nodes, so that, in case of multiple updates, the order of their execution is irrelevant.

[19] The encoding adopted in this proof is not necessarily the most efficient. E.g., compare the encoding of the operator *Board*(*x*,*y*,*z*) in Figure 10 with the simpler and more efficient one in Figure 8.

[20] For Theorem 1 to be valid, the setGraph planning notation must be extended with *negative* preconditions; this is necessary in order to represent the equivalent of a negative literal in the preconditions of a sentential operator. This can be done by adding a list Π of setGraphs to the analogical part of the preconditions of a setGraph operator and by requiring that, for the operator to be applicable in a state *s*, none of the setGraphs in Π be satisfiable in *s*.

[21] These negative goals can be easily transformed into positive ones; e.g., "¬`Needs(`*x*, *y*`)`" could be written as "`Has(`*x*, *y*`)`".

[22] In fact, in Example 5.3, suppose that the final destination of car A is Port$_1$. The optimal solution of the sentential sub-problem is to take A to Port$_4$; the resulting analogical problem would then require two other trips to take A from there to its final destination, resulting in a final plan containing three trips. The optimal plan for car A, however, would consist of taking it to Port$_3$ first, where it can refuel, and then to Port$_1$, where it would get the food and terminate.

[23] According to Definition 9, an action $a \in A$ of a domain is modelled as a function $\lambda \in \Lambda$ that maps models into models. Naturally, in order to be able to make the *process of reasoning about* (i.e., simulating) actions fully automatic, one must specify a general algorithm Γ that calculates the model $\lambda(M)$ for any given action model $\lambda \in \Lambda$ and any world model $M \in \Sigma$. For this to be possible, all functions in Λ will have to be (finitely) encoded as action *descriptions* (i.e., operators) so that they can be given as input to the procedure Γ.

[24] A set of operators containing conditional and quantified effects can be compiled into an equivalent set of containing only ground propositions (or ground setGraphs), using techniques similar to those of Gazen & Knoblock (1997) (see also (Fox & Long, 2003)). This is not possible, however, if the parameters of a setGraph operator contain numeric variables, as their instantiation would generate an infinite number of ground instances. To overcome this problem, any numeric node of the state appearing as a parameter in an operator should be replaced with an equivalent "primitive numeric expression" (e.g., node *x* in the *Board*(*x*,*y*,*z*) operator of Figure 8 could be replaced with `tot_cars(z)`), so that numbers can be manipulated only through their relationships with the objects of the domain and never appear as values to action parameters. This is, of course, the solution adopted in the sentential part of the representation, identical to that used by Fox & Long (2003) for PDDL2.1.

[25] Notice that *expressive equivalence* of two formalisms does not imply *equivalent efficiency*: i.e., the fact that an analogical language is as powerful as a sentential one does not imply that two encodings that they produce for the same problem can be solved with the same number of steps. This is true even if the search algorithm adopted for them is the same. Indeed, this was the set up for the experimental results considered in the third section of this chapter.

[26] See http://www-formal.stanford.edu/jmc/elaboration.html

References

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D. (ed.), Machine Intelligence, 3, 131–171. Edinburgh: Edinburgh University Press.

Barr, A., and Feigenbaum, E. A., eds. (1981). The Handbook of Artificial Intelligence, Vol. 1. Los Altos, California: William Kaufmann, Inc.

Barwise, J., and Etchemendy, J. (1995). Heterogeneous Logic. In (Glasgow *et al.*, 1995), chapter 7, pp. 211–234.

Barwise, J., and Etchemendy, J. (1998). Computers, visualization, and the nature of reasoning. In Bynum, T.W., & Moor, J.H. (eds.), The Digital Phoenix: How Computers are Changing Philosophy. Oxford: Blackwell Publishers.

Blum, A., & Furst, M.F. (1997). Fast planning through planning graph analysis. Artificial Intelligence, 90, 281–300.

Bonet, B., & Thiébeaux, S. (2003). GPT meets PSR. In Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling, 102–112. AAAI Press.

Cesta, A., & Oddi, A. (1996). DDL.1: A formal description of a constraint representation language for physical domains. In Ghallab, M., & Milani, A. (eds.), New Directions in AI Planning. 341–352. Amsterdam, NL: IOS Press.

Chang, C., & Keisler, H. (1977). Model Theory. New York: Elsevier Press.

Cook, S., & Liu, Y. (2003). A Complete Axiomatization for Blocks World. Journal of Logic and Computation, 13(4), 581–594.

Davidson, M., & Garagnani, M. (2002). Pre-processing planning domains containing Language Axioms. In Grant, T., & Witteveen, C. (eds.), Proceedings of the Twenty-first Workshop of the UK Planning and Scheduling SIG, Delft (NL), November 2002. 23–34.

Dimopoulos, Y., Nebel, B., & Koehler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In Steel, S., & Alami, R. (eds.), Recent Advances in AI Planning (Lecture Notes in Computer Science, 1348). 169–181. Springer Verlag.

Dretske, F.L. (1981). Knowledge and the Flow of Information. Cambridge, MA: MIT Press.

Erdem, E., & Lifschitz, V. (1999). Transformations of logic programs related to causality and planning. In Gelfond, M., Leone, N., & Pfeifer, G. (eds.), Logic Programming and Nonmonotonic Reasoning: Proceedings of the Fifth International Conference (Lecture Notes in Artificial Intelligence, 1730), 107–116. Springer Verlag.

Erdem, E., & Lifschitz, V. (2003). Tight Logic Programs. Special Issue of Theory and Practice of Logic Programming on Programming with Answer Sets, 3(4–5), 499–518.

Ernst, M., Millstein, T., & Weld, D. (1997). Automatic SAT-compilation of planning problems. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), 1169–1177. San Francisco, California: Morgan Kaufmann.

Fikes, R.E., & Nilsson, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2, 189–208.

Forbus, K. (1995). Qualitative Spatial Reasoning: Framework and Frontiers. In (Glasgow *et al.*, 1995), chapter 6, pp. 183–202.

Forbus, K., Nielsen, P., & Faltings, B. (1987). Qualitative Kinematics: A Framework. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI'87), 430– 436. San Francisco, California: Morgan Kaufmann.

Forbus, K., Nielsen, P., & Faltings, B. (1991). Qualitative Spatial Reasoning: The CLOCK Project. Artificial Intelligence, 51(1-3), 417– 471.

Fox, M., & Long, D. (2001). STAN4: A Hybrid Planning Strategy Based on Sub-problem Abstraction. AI Magazine, 22(3), 81–84.

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.

Garagnani, M. (2003). Model-based Planning in Physical domains using SetGraphs. In Coenen, F., Preece, A., & Macintosh, A. (eds.), Research and Development in Intelligent Systems XX (Proceedings of AI-2003). 295–308. Springer Verlag.

Garagnani, M. (2004). A Framework for Planning with Hybrid Models. In Proceedings of ICAPS’04 Workshop on Connecting Planning Theory with Practice, Whistler (Canada). 14–19.

Garagnani, M., & Ding, Y. (2003). Model-based Planning for Object-rearrangement Problems. In Proceedings of ICAPS’03 Workshop on PDDL, Trento (Italy). 49–58.

Gazen, B.C., & Knoblock, C.A. (1997). Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In Steel, S., & Alami, R. (eds.), Recent Advances in AI Planning (Lecture Notes in Computer Science, 1348). 221–233. Springer Verlag.

Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. Journal of Logic Programming 17, 301–322.

Georgeff, M.P. (1987). Planning. Annual Review of Computer Science, 2, 359–400.

Giunchiglia, E., & Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), 623–630. AAAI Press.

Glasgow, J., & Malton, A. (1994). A Semantics for Model-Based Spatial Reasoning. Technical Report 94-360, Department of Computing and Information Science, Queen’s University, Kingston, Ontario.

Glasgow, J., Narayanan, N.H., & Chandrasekaran, B., eds. (1995). Diagrammatic Reasoning. Menlo Park (CA), Cambridge (MA), London (England): AAAI Press / The MIT Press.

Halpern, J.Y., & Vardi, M.Y. (1991). Model Checking vs. Theorem Proving: A Manifesto. In Allen, J., Fikes, R., & Sandewall, E. (eds.), Principles of Knowledge representation and Reasoning: Proceedings of the Second International Conference (KR-91), 325–332. Morgan Kaufmann.

Han, J. (1989). Compiling general linear recursions by variable connection graph analysis. Computational Intelligence, 5, 12–31.

Harel, D. (1988). On Visual Formalisms. Communications of the ACM, 13(5), 514–530.

Haslum, P., & Geffner, H. (2000). Admissible Heuristics for Optimal Planning. In Chien, S., Kambhampati, S., & Knoblock, C. (eds.), Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, 140–149. AAAI Press.

Hayes, P.J. (1985). Some problems and non-problems in representation theory. In Brachman, R., & Levesque, H. (eds.), Readings in Knowledge Representation. 4–22. Los Altos, CA: Morgan Kaufmann.

Hayes, J.R., & Simon, H.A. (1977). Psychological differences among problem isomorphs. In Castellan, N., Pisoni, D., & Potts, G. (eds.), Cognitive Theory. Erlbaum.

Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research, 14, 253–302.

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In Neumann, B. (ed.), Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI’92), 359–363. Chichester, England: John Wiley and Sons.

Kautz, H., & Selman, B. (1998). The role of domain-specific knowledge in the planning as satisfiability framework. In Simmons, R., Veloso, M., & Smith, S. (eds.), Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, 181–189. AAAI Press.

Kautz, H., & Selman, B. (1999). Unifying SAT-based and Graph-based planning. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 318–325. Morgan Kaufmann.

Khardon, R., & Roth, D. (1996). Reasoning with Models. Artificial Intelligence, 87, 187–213.

Koedinger, K.R. (1992). Emergent properties and structural constraints: Advantages of diagrammatic representations for reasoning and learning. In Narayanan, N.H. (ed.), AAAI Spring Symposium on Reasoning with Diagrammatic Representations: Working Notes. 151–156. Menlo Park, California: AAAI Press.

Kulpa, Z. (1994). Diagrammatic Representation and Reasoning. Machine GRAPHICS & VISION, 3(1/2), 77–103.

Larkin, J.H., & Simon, H.A. (1987). Why a Diagram Is (Sometimes) Worth Ten Thousand Words. Cognitive Science 11, 65–99. Reprinted in (Glasgow *et al.*, 1995), chapter 3, 69–109.

Levesque, H. (1986). Making believers out of computers. Artificial Intelligence, 30, 81–108.

Lehmann, F. (1992). Semantic Networks. Computers and Mathematics with Applications, 23(2-5), 1–50.

Lifschitz, V. (1990). On the semantics of STRIPS. In Allen, J., Hendler, J., & Tate, A. (eds.), Readings in Planning. 523–530. San Mateo, California: Morgan Kaufmann.

Lifschitz, V. (2002). Answer set programming and plan generation. Artificial Intelligence, 138, 39–54.

Lindsay, R.K. (1995). Images and Inference. In (Glasgow *et al.* 1995), chapter 4, pp.111–135.

Long, D., & Fox, M. (2000). Automatic synthesis and use of generic types in planning. In Chien, S., Kambhampati, S., & Knoblock, C. (eds.), Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, 196–205. AAAI Press.

McAllester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), 634–639. AAAI Press.

McCarthy, J. (1958). Programs with common sense. In Proceedings of the Symposium on the Mechanization of Thought Processes, Vol. 1, 77–84. Reprinted in Brachman, R., & Levesque, H. eds., (1985) Readings in Knowledge representation. 299–307. Los Altos, California: Morgan Kaufmann.

McCarthy, J., & Hayes, P.J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In In Allen, J., Hendler, J., & Tate, A. (eds.), Readings in Planning. 393–435. San Mateo, California: Morgan Kaufmann.

McDermott, D., Knoblock, C., Veloso, M., Weld, S.D., & Wilkins, D. (1998). PDDL - the planning domain definition language. Version 1.2. Technical Report, Department of Computer Science, Yale University (CT).

More, T. (1981). Notes on the diagrams, logic and operations of array theory. In Bjorke, O., & Franksen, O. (eds.), Structures and Operations in Engineering and Management Systems. 497–666. Trondheim, Norway: Tapir Publishing.

Myers, K., & Konolige, K. (1995). Reasoning with analogical representations. In (Glasgow *et al.*, 1995), chapter 9, pp. 273–301.

Palmer, S.E. (1978). Fundamental Aspects of Cognitive Representation. In Rosch, E., & Lloyd, B.B. (eds.), Computing and Categorization. 259–303. Hillsdale, NJ: Earlbaum.

Penberthy, J. S., & Weld, S. D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In Nebelm B., Rich, C., & Swartout, W.R. (eds.), Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR-92), 108–114. Morgan Kaufmann.

Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R., Levesque, H.J., Reiter, R. (eds.), Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89), 324–332. Morgan Kaufmann.

Petri, C.A. (1963). Fundamentals of a Theory of Asynchronous Information Flow. In Proceedings of IFIP Congress 62, 386–390. Amsterdam: North Holland.

Pollock, J.L. (1998). Perceiving and Reasoning about a Changing World. Computational Intelligence 14(4), 498–562.

Shanahan, M. (1997). Solving the Frame Problem. Cambridge, MA: MIT Press.

Simon, H.A. (1981). The Sciences of Artificial. (2nd ed.) Cambridge, MA: MIT Press.

Sloman, A. (1975). Afterthoughts on analogical representations. In Proceedings of the First Workshop on Theoretical Issues in Natural Language Processing (TINLAP-1), Cambridge, MA. 164–171.

Sowa, J.F. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, MA: Addison-Wesley.

Sussman, G.J. (1990). The Virtuous Nature of Bugs. In Allen, J., Hendler, J., & Tate, A. (eds.), Readings in Planning. 111–117. San Mateo, CA: Morgan Kaufmann.

Swoboda, N., & Allwein, G. (2002). A case study of the design and implementation of heterogeneous reasoning systems. In Magnani, L., Nersessian, N.J., & Pizzi, C. (eds.), Logical and Computational Aspects of Model-Based Reasoning. 3–20. Dordrecht, NL: Kluwer.

Thiébeaux, S., Hoffmann, J., & Nebel, B. (2003). In Defense of PDDL Axioms. In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico. Morgan Kaufmann.

Weld, D.S. (1999). Recent Advances in AI Planning. AI Magazine, 20(2), 93–123.

Wilkins, D.E., & desJardins, M. (2001). A call for Knowledge-based Planning. AI Magazine, 22(1), 99–115.