Technical Report No: 2003/15

Planning with Hybrid and Analogical Representations

Max Garagnani

18th November, 2003

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

The Open University

Planning with Analogical and Hybrid Representations*

Max Garagnani
Department of Computing, The Open University
M.Garagnani@open.ac.uk

This report illustrates how new methods and techniques from the area of knowledge representation and reasoning can be adopted and exploited in planning to produce new, more efficient domain-description languages.

Planning domain description formalisms should be expressive and customisable, and yet be able to produce domain encodings that allow the planner to concentrate all of the computational effort on the *search* for a solution, rather than on calculating the trivia of the problem. This paper argues that most of the modern, 'sentential' domain-modelling languages do not meet the latter requirement, and, when applied to realistically complex domains, produce encodings that are subject to the inefficiencies of the *ramification* problem.

The solution proposed here consists of adopting *non*-sentential domain representations in planning. In particular, recent experimental evidence (Garagnani and Ding 2003) indicates that the adoption of *analogical* descriptions can lead to significant planning speed-ups. However, although more efficient, when compared to sentential languages these representations tend to be less expressive and less universally applicable. The main aim of this paper is to provide a theoretical basis for *hybrid* planning, in which analogical and sentential domain representations can be seamlessly integrated and used interchangeably, thereby overcoming the limitations and exploiting the advantages of both paradigms.

The first part of the document clarifies the nature of analogical representations and illustrates how they can avoid the frame and the ramification problems, which afflict all sentential languages. The central section presents an expressive analogical representation (based on the structure of "setGraph") and a formal framework for hybrid planning built on it, while the last section discusses related work, limitations and future directions.

1 Sentential vs. Analogical domain representation

During the last decade, the need for new formalisms able to support common-sense reasoning more efficiently than the widely adopted *sentential* (or propositional, Fregean (Kulpa 1994)) representation has lead to a resurgence of interest in 'nonlinguistic' types of knowledge representation, also referred to as *diagrammatic* (Larkin & Simon 1987), *analogical* (Sloman 1975) (Dretske 1981) (Myers & Konolige 1995), *homomorphic* (Barwise & Etchemendy 1995), *direct* (Levesque 1992) and *model-based* (Barr & Feigembaum 1981) (Halpern & Vardi 1991) representations. In short, a sentential representation is one in which the current world-state is described by a set of well-formed formulæ (sentences) of a formal language (e.g., propositional or predicate logic). While a sentential representation can be said to *describe* the domain represented, in an analogical representation the world is *modelled* by a description that is structurally similar to the represented domain (Kulpa 1994). Barwise and Etchemendy concisely describe the main difference between the two representations:

> "[…] *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,*

* This work was partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC), grant no. GR/R53432/01.

whereas the corresponding mapping in the case of linguistic representations is anything but structure preserving." (Barwise & Etchemendy 1995, p.214)

Recently, heterogeneous (or hybrid) models (Barwise & Etchemendy 1995, 1998) (Swoboda & Allwein 2002) have also been explored, 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.

In spite of these recent developments in knowledge representation and reasoning, planning research has remained fundamentally sentential, failing to assimilate and exploit the results of these closely related areas of AI. Most current planning domain-description languages continue to encode all aspects of a problem (including *spatial* and *topological* relations) using exclusively propositional representations. Although these languages are very general, they produce representations that are often *inefficient*. This is argued in more details below.

1.1 Planning with propositional languages

For many years, researchers in AI have regarded the frame problem (McCarthy & Hayes 1969) as presenting a major difficulty for reasoning about action (Georgeff 1987) (Shanahan 1997). The frame problem (in essence, having to specify, for each action, which properties of the world remain *unaffected* by its execution) can be seen as the result of adopting a specific knowledge representation paradigm, and was addressed in STRIPS (Fikes & Nilsson 1971) by requiring that the representation of an action (operator) explicitly listed only those propositions that *are* affected by the actual execution of the represented action, while those that are unaffected are simply assumed to persist (Lifschitz 1986). This assumption, still lying at the basis of modern planning domain description languages (Fox & Long 2003), allows the frame problem to be avoided, but does not address a second, equally serious representational issue: the so-called *ramification* problem (Georgeff 1987). John Pollock (Pollock 1998) accurately described 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.* […] [I]*n the real world, all actions have infinitely many ramifications stretching into the indefinite future. This is a problem for reasoning about change deductively* […]" (Pollock 1998, p.536)

In order to understand the actual impact of this problem on planning, let us consider a simple variation of the Blocks-World (BW) domain, in which the robot arm can lift towers of m blocks, with $m \leq h \in \aleph$. In order to decide which blocks may be lifted in a given state, it should be possible to augment the problem description with the two following domain axioms:

$(\alpha_1)\ \forall\, x,y : \mathrm{On}(x, y) \rightarrow \mathrm{Above}(x, y, 1)$
$(\alpha_2)\ \forall\, x,z\ (\exists\, y,n : \mathrm{On}(x, y) \wedge \mathrm{Above}(y, z, n) \rightarrow \mathrm{Above}(x, z, n+1)$

The condition "$\mathrm{Clear}(x) \wedge \mathrm{Above}(x, y, n) \wedge (n \leq h)$" would then suffice to guarantee that y is the lowest block of a 'liftable' tower. Unfortunately, no one has yet provided a semantics for domain axioms containing numeric fluents, such as those shown above. Most importantly, given a certain world state (described using exclusively 'On( )' expressions), deducing the 'Above(x,y,n)' relations for all the possible blocks requires a number of axiom applications that grows exponentially in the size (arity) of the axioms. More precisely, if 'o' is the number of objects (constant symbols) of the domain and 'r' is the arity of the axioms, the number of deductions required is in the order of $k \cdot o^r$, for a certain constant k.

Because of this, extending planners to deal with domain axioms seems likely to involve significant additional computational costs. Consider, for instance, how a forward state-space

planner could solve a problem in the above BW domain with axioms (α_1) and (α_2): whenever the truth value of any of the 'On( )' propositions changes, it will be necessary for the planner to re-calculate all of the 'Above( )' relations, as the truth of some of them will have been affected by the change. A backward-search planner would incur in similar problems. Consider, for example, the process of determining how to achieve the goal (or precondition) 'Above(x,y,j)', in which the variables x and y are still unbound (i.e., how to build a tower of j+1 blocks, for a given j, using any set of blocks). Transforming this expression into the equivalent formula "On(x, z_1) $\wedge$ On(z_1, z_2) $\wedge$ … $\wedge$ On(z_{j-1}, y)" (which can then be achieved using the standard BW operators) requires a rather complex process of reasoning. In addition, in a backward-chaining search, the presence of axioms would significantly increase the branching factor. Several researchers (e.g., (Garagnani 2000)(Gazen & Knoblock 1997) (Davidson & Garagnani 2002) (Thiébeaux, Hoffman & Nebel 2003)) have tackled this problem by adopting a pre-processing approach, and have explored the possibility of compiling axioms away in the domain description. However, recent theoretical results (Thiébeaux *et al.* 2003) show that this approach leads (in general) to exponentially longer descriptions and plans. Experimental evidence suggests that the resulting performances can be even worse than those obtained with planners that are able to deal with domain axioms internally (Thiébeaux *et al.* 03).

In summary, the ramification problem and the presence of domain axioms, intrinsically related, add to the planning process a significant computational overhead, which may become severe or even unacceptable if planning is to be carried out in realistically-complex scenarios. In fact, when reasoning about (i.e., simulating) action in real-world domains, innumerable physical properties of the world should be taken into account, such as sound and light propagation, temperature, gravity, fluid and gas dynamics. Using Pollock's original example, among the effects of striking a match we must 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, making that match unavailable for future use, etc.*" (Pollock 1998, p.537).

In realistic domains, the number and complexity of the axioms would quickly become overwhelming, making planning very difficult and inefficient.

The problem of domain axioms and that of frame axioms (requiring, respectively, the explicit specification of all properties that are – and that are not – affected by action) are, ultimately, two facets of the same representational problem, which lies in the use of purely sentential domain description languages. Being purely descriptive, propositional planning languages are quite flexible and expressive. However, because of their descriptive nature, they require *all* properties and constraints of the world (even the most trivial, such as the fact that an object cannot be moved on top of itself) to be represented *extrinsecally* (Palmer 1978), i.e., to be explicitly imposed on the model through the addition of supplementary elements, in the form of formulæ and axioms. As seen earlier, this can quickly lead to a combinatorial explosion in the number of trivial inferences that must be explicitly represented and carried out – namely, to the ramification problem. Hence, although propositional languages can be very expressive, they produce inefficient representations for domains that involve a large number of distinct entities subject to (even simple) physical constraints. In order to overcome these problems, more adequate planning domain description languages are needed.

1.2 Planning with Analogical Representations

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) (Kulpa 1994) (Levesque 1992) (Khardon 1996)). In particular, analogical and diagrammatic representations have long been of interest to the knowledge representation community (Amarel 1968) (Sloman 1975) (Hayes 1974) (Myers and Konolige 1995) (Lindsay 1995) (see (Glasgow, Narayanan & Chandrasekaran 1995) for a useful collection). In order to characterise more accurately such representations, let us introduce a simple example of analogical representation for the familiar Blocks-World (BW) domain.

Consider, for instance, a representation of BW in which the state is modelled as a set of lists of characters. Each list denotes a (possibly empty) stack of blocks, and each character represents a block. For example, in a four-block problem, the state S = {[A,B,C],[D],[ ],[ ]} would correspond to the propositional description {On(C,B), On(B,A), Clear(C), Clear(D)}, in which predicates have their standard meaning (empty lists denote empty spaces on the table). The specification (using an appropriate syntax) of a generic operation for transforming legal states into legal states (consisting of removing the last character of a non-empty list and appending it to another list) completes the description of the domain model.

Using this simple example of analogical representation for the (one-operator) BW domain it is already possible to identify the main features that differentiate such representations from propositional ones. In a propositional representation the objects of the domain are represented as constant symbols (e.g., A, B, C,... etc.). The relevant relationships (like 'on' and 'above') between objects are described as sets of associations (relational instances) between such symbols, and are identified using other symbols (predicates) of the language (e.g., "On(B,A)", "Above(C,A)"). The specific syntax chosen to build such formulæ, however, has no bearing to the properties of the represented world. In particular, the properties of the relations that exist between the objects of the domain and their interactions are *imposed* on the formal model through the specification of axioms and logical rules (e.g., axioms (α_1) and (α_2)). By contrast, in an analogical representation the objects of the domain and the relationships that exist between them are not described by sets of relational instances, but modelled using appropriate data structures. The *syntax* of such data structures mirrors, for the relevant aspects, the semantics (relations and properties) of the problem domain (Barr & Feigembaum 1981, pp.200-206). In particular, the relations between elements of the representing structures of the model and the corresponding represented relations of the domain have the same *algebraic* structure (using Palmer's term, they are "naturally isomorphic" (Palmer 78)). In addition, relations between objects of the domain do not need to be *explicitly* declared in the model and appear as 'pointable' symbols of the description.

To wit, consider the BW domain example. The analogical, list-based representation consists of a set of formulæ having the following syntax:

$$[\ arg_1, arg_2, \ldots, arg_n]$$

This syntax clearly reflects the semantics of BW: the first character of the list (arg_1) always represents the lowest block of a stack, while two consecutive characters arg_k, arg_{k+1} indicate that block arg_{k+1} is on block arg_k. The rightmost character of a list denotes a 'clear' block. In contrast, the formulæ used in the propositional representation adopt the following syntax:

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

This syntax has no direct relation with the structure and properties of the BW domain.

In addition, consider the spatial relation "above", represented in the model by the relation "to the right of", defined on the (linearly ordered) characters of a list[1]. The transitive property of the relation "above" (originally imposed on the model through the addition of axioms (α_1)

[1] More precisely, the block denoted by character *x* is *above* the one denoted by *y* iff character *x* appears to the right of *y* (in the same list).

and (α_2)) is an *implicit* property of the relation "to the right of", and does not need to be explicitly imposed or accounted for during the reasoning process. In other words, the latter relation and the represented spatial relation "above" are naturally isomorphic (Palmer 1978), or, to use Koedinger's terminology, the transitivity of the relation "to the right of" is an *emergent property* of the representation (Koedinger 1992).

Thanks to these features, analogical descriptions can implicitly embody *constraints* that sentential representations would normally have to make explicit: the relevant properties of the relations existing between the objects of the domain are *inherent* to the structure of the representing relations, and do not need to be explicitly imposed on the model or included in the description (Myers & Konolige 1995). The presence of such "inherent constraints" (Palmer 1978) reduces the computational complexity of inference, avoiding the combinatorial explosion of trivial deductions that would have to be explicitly represented in sentential reasoning systems, and easing the frame and ramification problems (cf. (Lindsay 1995, p.112)). In the BW example, given a certain state description, the problem of deducing – using axioms (α_1) and (α_2) – whether a tower is 'liftable' becomes one of simply checking the current model, by counting the number of characters that follow a given one in a list. This check can be carried out in time *linear* in the number of blocks. In other words, thanks to the *model-based* (Winslett 1988) nature of the representation, the computationally expensive theorem-proving aspects of the planning process can be replaced by simple model-checking.

The idea of replacing theorem proving (which, even in the propositional case, is co-NP-Hard) with model checking is not new (e.g., see (Halpern & Vardi 1991) (Khardon 1996)), and has been successfully applied to planning in the past, leading to the framework of planning as model-checking (Giunchiglia & Traverso 1999). In the latter approach, however, the semantic models created and checked are not *analogical* models of the world but state-transition structures encoding all the possible global transitions. Although very general, this representation fails to capture implicitly the inherent structure of the domain, and hence to avoid the ramification problem that crops up in complex, real-world domains.

The main difficulty with analogical and model-based representations lies in finding a sufficiently general semantic model that can appropriately represent all complex aspects of the world. In fact, due to the implicit, unalterable structure of the relations that they employ, analogical representations are less expressive than propositional ones. Indeed, in the real world, only few, simple domains can be encoded using *purely* analogical models, while most problems appear to require *heterogeneous* (Barwise & Etchemendy 1995) formalisms, combining analogical and propositional representations. Aiming to address these issues, the following section proposes (1) a new, general analogical structure (setGraph) sufficiently expressive to encode any PDDL2.1 (level 2) domain, and (2) a theoretical framework for *hybrid* planning, in which domain descriptions containing qualitative, quantitative, sentential and analogical features, can be integrated and used interchangeably.

2 A Theoretical framework for Analogical and Hybrid Planning

Following (Lifschitz 1986) and (Fox & Long 2003), the real world is taken to be, at any instant of time, in a certain *state* (an element of a set 'S' of possible ones). A domain is a subset of the real world consisting of a finite set *I* of *entities* and finite sets of relations among and properties of entities. To describe the current world state, we use a 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* (i.e., non-numeric), 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 *sort hierarchy* that organises all symbols of C into subsets (sorts) $D_1, \ldots, D_k$ such that $(\cup_{i\in\{1,\ldots k\}} D_i) = C$. The *i*-eth argument '$t_i$' of a logical relation (function) symbol *p* (*f*) will be required to be of sort D_p^i (D_f^i) by imposing $t_i \in D_p^i$ ($\in D_f^i$). The *wff* of a many-sorted language, *logical* and *numeric atoms*, are built as follows:

Definition 1 – *Given a many-sorted language* $\mathcal{L}=\langle P,F,C\rangle$ *with* $C = \cup_{i\in\{1,\ldots k\}} D_i$,

- *t is a* term *iff* $t\in C$; *x is a* number *iff* $x\in\Re$, *where* $\Re$ *is the set of real numbers;*
- *if* $f\in F$ *is an m-placed logical function symbol, then* $f(t_1,\ldots, t_m)$ *is a* primitive numeric expression (PNE) *iff* $t_1, \ldots, t_m$ *are terms and,* $\forall i\in\{1,\ldots, m\}$, $t_i\in D_f^i$;
- *a number is a* numeric expression (NE); *if* $h\in F$ *is an m-placed numeric function symbol, then* $h(t_1, \ldots t_m)$ *is a* numeric expression *iff* $t_1, \ldots, t_m$ *are either PNEs or NEs;*
- *if* $p\in P$ *is an n-placed logical relation symbol, then* $p(t_1,\ldots, t_n)$ *is an* atomic formula *(or* logical atom*) iff* $t_1,\ldots t_n$ *are terms and,* $\forall i\in\{1,\ldots, n\}$, $t_i\in D_p^i$;
- *if* $q\in P$ *is an n-placed numeric relation symbol, then* $q(t_1,\ldots, t_n)$ *is a* numeric atomic formula *(or* numeric atom*) iff* $t_1,\ldots, t_n$ *are either PNEs or NEs.*

Each symbol of $\mathcal{L}$ is given the standard semantics (interpretation) in the domain of interest (Chang and Keisler 1977, Section 1.3). In particular, we assume that an interpretation function *g* maps each constant symbol $c\in C$ to an entity $g(c)=a\in A$, each *m*-placed logical function symbol $f\in F$ to a function $g(f) = f'$: $A^m \rightarrow \Re$, and each *n*-placed logical relation symbol $p\in P$ to a relation $g(p) = p' \subset A^n$. 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, unlike the latter (fixed) functions and relations, which are predetermined and do not change, the value and truth of $f'(a_1,\ldots, a_m)$ and $p'(a_1,\ldots, a_n)$ may depend on the current state.

Given a real domain, let S be the set of possible states in which the world can be. The interpretation function allows to determine, for each state *s*, which atoms of $\mathcal{L}$ are *satisfied* in this state, and the value (in *s*) of each PNE and NE of $\mathcal{L}$:

Definition 2 – *Given a language* $\mathcal{L} = \langle P,F,C\rangle$*, a state* $s\in S$ *and an interpretation function* '*g*'*, an atom* $p(t_1,\ldots, t_n)$ *of* $\mathcal{L}$ *is* satisfied *in s iff* $g(t_1),\ldots, g(t_n)$ *share relation g(p) in state s. If argument* t_i *is a number, then* $g(t_i) = t_i$*. If* $t_i = f(t_{i1},\ldots t_{i\,m})$ *– i.e., argument* t_i *is a PNE or a NE – then* $g(t_i)$ *is the value (in s) of the function g(f) calculated in* $(g(t_{i1}),\ldots g(t_{i\,m}))$.

In what follows we assume that, for a given domain and language $\mathcal{L}$, a *fixed* interpretation function *g* is adopted; the value $g(\, f(t_1,\ldots t_m)\,)$ in a state *s* will be denoted as $f(t_1,\ldots t_m)|_s$.

Consider a generic data structure D (such as a tree, a list, a graph, an array) containing elements taken from a possible universe U (like characters, strings, numbers, and so forth). Let $\mathcal{D}$ be the set of all the possible (finite) legal instances of D that can be generated using the elements in U, and let $\Re_\perp = \Re \cup\{\perp\}$, where $\perp$ denotes the undefined value. The elements of $\Re_\perp$ will be called *R-values*. A "domain representation structure" for a domain with language $\mathcal{L}$ consists of a (set of instances of a) data structure and a set of procedures for inspecting such data structure:

Definition 3 – *A* domain representation structure *for a language* $\mathcal{L}=\langle P,F,C\rangle$ *with interpretation g is a triple* $\langle\Psi, \Phi, \Delta\rangle$*, where* $\Delta (\subseteq \mathcal{D})$ *is a set of instances of a data structure D (containing elements of U) and* Ψ, Φ *are sets of procedures such that:*

- *each n-placed logical relation symbol $p \in P$ is associated to a procedure $\psi_p \in \Psi$ such that $\psi_p: \Delta \times U^n \rightarrow \{True, False\}$;*
- *each m-placed logical function symbol $f \in F$ is associated to a procedure $\phi_f \in \Phi$ such that $\phi_f: \Delta \times U^m \rightarrow \Re_\perp$;*
- *each n-placed numeric relation symbol $q \in P$ is associated to a procedure $\psi_q \in \Psi$ such that $\psi_q: (\Re_\perp)^n \rightarrow \{True, False\}$ and $\psi_q(x_1,\ldots,x_n)=True$ iff $(x_1,\ldots,x_n) \in g(q)$ (if $\exists\, x_i$ such that $x_i = \perp$, then ψ_q returns 'False');*
- *each m-placed numeric function symbol $h \in F$ is associated to a procedure $\phi_h \in \Phi$ such that $\phi_h: (\Re_\perp)^n \rightarrow \Re_\perp$ and $\phi_h(x_1,\ldots,x_m)$ calculates $g(h)(x_1,\ldots,x_m) \in \Re$ (returning $\perp$ if $g(h)(x_1,\ldots,x_m)$ is undefined or if $\exists\, x_i$ such that $x_i = \perp$).*

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 world state.

Definition 4 – *Given a language $\mathcal{L}=\langle P,F,C\rangle$ with domain representation structure $\mathcal{R} = \langle \Psi,\Phi,\Delta\rangle$, a* model in $\mathcal{R}$ *is a pair $M=(d,\varepsilon)$ such that $d \in \Delta$ is an instance of a data structure D and $\varepsilon: C \rightarrow U$ is a 1-1 mapping from symbols of C to elements of the universe U.*

Definition 5 – *Given a domain representation structure $\mathcal{R} = \langle \Psi,\Phi, \Delta\rangle$ for a language $\mathcal{L}$, a model $M=(d,\varepsilon)$ in $\mathcal{R}$* represents *a state $s \in S$ (written "$M \cong_{\mathcal{R}} s$") iff, for any 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*;
- $\phi_f(d,\varepsilon(t_1),\ldots,\varepsilon(t_m)) = f(t_1,\ldots t_m)|_s$ *, and if* $f(t_1,\ldots t_m)|_s$ *is undefined, then* $\phi_f(d, \varepsilon(t_1),\ldots,\varepsilon(t_m))=\perp$

Proposition 1 – *Given a state $s \in S$, a domain representation structure $\mathcal{R} = \langle \Psi,\Phi, \Delta\rangle$ for a language $\mathcal{L}$ and a model $M = (d,\varepsilon)$ in $\mathcal{R}$, if $M \cong_{\mathcal{R}} s$ then, for all numeric atoms $q(t_1,\ldots,t_n)$ of $\mathcal{L}$,*

$$\psi_q(\, e(t_1),\ldots, e(t_n)\,) = True \;\; iff \;\; q(t_1,\ldots, t_n) \text{ is satisfied in } s,$$

where 'e' is an evaluation function such that

- *if t_j is a number, then $e(t_j) = t_j$;*
- *if $t_j = f(t_{j1},\ldots t_{j\,m})$ is a PNE, then $e(t_j) = \phi_f(\, d, \varepsilon(t_{j1}),\ldots,\varepsilon(t_{j\,m})\,)$;*
- *if $t_j = f(t_{j1},\ldots t_{j\,m})$ is a NE, then $e(t_j) = \phi_f(\, e(t_{j1}),\ldots, e(t_{j\,m})\,)$.*

The proof follows directly from Definitions 2, 3 and 5.

Example 1 – Consider the Blocks World domain, in which a set of blocks and a table are the entities of interest, "*to be above*" is the relevant relation between entities, and the weight of a block is the only property of interest. The language

$$\mathcal{L}_1 = \langle P_1,F_1,C_1\rangle = \langle \{\text{Above}, \geq\}, \{\text{Weight}, +, -, *, /\}, \{T, B_1, B_2, B_3\}\rangle,$$

with sorts *Block* $=\{B_1, B_2, B_3\}$ and *Table* $=\{T\}$ 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*, "Above" is a 2-placed logical relation symbol with unrestricted argument type. "$\geq$" is a 2-placed numeric relation symbol, and '+', '–', '$\cdot$' and '/' are 2-placed numeric function symbols. The interpretation of the symbols of $\mathcal{L}_1$ is intuitive: 'Weight' denotes the function

Block $\rightarrow \Re$ returning the weight of a block, '$\geq$' is the binary relation "greater than or equal to" on $\Re$, and '+', '–', '*', '/' are the standard arithmetic operations on $\Re$. 'Above' is mapped to the "*above*" relation between objects (blocks and table).

Consider a domain representation structure based on a data structure D composed of a set S of strings and of an array W (of length = 3) of *R-values* (values in $\Re_\perp$); the universe U of strings that can appear in S includes all the constant symbols of $\mathcal{L}_1$, {"T", "B_1", "B_2", "B_3"}= U_1, and all the logical atoms of $\mathcal{L}_1$. $\psi_{Above}(d, x, y)$ is a procedure that takes as input an instance d=(S,W) of the data structure D and two strings $x, y \in U_1$ and returns '*True*' iff the string "Above(x, y)" appears in S (where x, y have been replaced with the corresponding strings). $\psi_{\geq}(x, y)$ is a procedure that returns '*True*' iff the real number x is equal to or greater than the real number y (if $x = \perp$ or $y = \perp$, $\psi_{\geq}(x, y)$ returns '*False*'). $\phi_{Weight}(d, c)$ is a procedure that takes as input a model d=(S,W) and an element $c \in U_1$ and returns the value of W(0) if c="B_1", of W(1) if c="B_2", of W(2) if c="B_3", and $\perp$ otherwise. The procedures ϕ_+, ϕ_-, ϕ_* and $\phi_/$ take two R-values $\in \Re_\perp$ and return the result of the corresponding arithmetic operation applied to the input (returning $\perp$ if the result is undefined, or if any of the inputs is = $\perp$).

Then, given a model M = (d,ε) such that $\varepsilon: C_1 \rightarrow U_1$ maps constant symbols of $\mathcal{L}_1$ to equivalent strings, M represents a state s of the domain iff set S (in d) contains all and only the logical atoms of $\mathcal{L}_1$ which are satisfied in s, and cells W(0), W(1), W(2) of the array W contain the values corresponding to the weights of the three blocks of the domain (or '$\perp$' if the weight is unknown). Thanks to Proposition 1, if M represents state s, procedure $\psi_{\geq}$ can be used to determine whether any arbitrarily-complex numeric atom of $\mathcal{L}_1$ is satisfied in s – for example, whether $\geq$ (/(*(Weight(B_2),2.5),Weight(B_1)), Weight(B_1)) is satisfied.
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 W(i–1) set to $\perp$, so that the PNE 'Weight(x)' is evaluated = $\perp$ if x does not exist.

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

Definition 6 – *A real-world* 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 set of total functions* $a: S \rightarrow S$.

Definition 7 – *Given a real domain* $Dom=\langle S,A \rangle$*, an associated language* $\mathcal{L}$ *with domain representation structure* $\mathcal{R}$ *, and a set of models* Σ *in* $\mathcal{R}$*,* Σ represents S *iff there exists a bijective function* $\rho: \Sigma \rightarrow S$ *such that,* $\forall M \in \Sigma$, $M \cong_{\mathcal{R}} \rho(M)$.

In other words, a set of models Σ represents a set of states S iff each model in Σ represents exactly one state in S, and each state of S has exactly one model in Σ that represents it. Given a set of models Σ representing the states S, any action a: S$\rightarrow$S can be modelled as a function $\alpha: \Sigma \rightarrow \Sigma$ which transforms (corresponding) model M into (corresponding) model M':

Definition 8 – *Given a domain* $Dom=\langle S,A \rangle$ *with language* $\mathcal{L}$*, a domain representation structure* $\mathcal{R}$ *for* $\mathcal{L}$ *and a set of models* Σ *representing S, a function* $\alpha: \Sigma \rightarrow \Sigma$ *is a* sound model *of an action* $a: S \rightarrow S$ *iff, for any model* $M \in \Sigma$ *and state* $s \in S$ *such that* $M \cong_{\mathcal{R}} s$, $\alpha(M) \cong_{\mathcal{R}} a(s)$.

Thus, a sound model of an action *a* maps each world model M (representing a state *s*) into the model M′ which represents the state resulting from the application of action *a* to *s*.

Definition 9 – *A* domain model *for a domain* $\langle S, A\rangle$ *is a quadruple* $\langle \mathcal{L}, \mathcal{R}, \Sigma, \Lambda\rangle$*, consisting of a language* $\mathcal{L}$*, a representation structure* $\mathcal{R}$ *for* $\mathcal{L}$*, a set of models* Σ *(adopting representation structure* $\mathcal{R}$*) representing S and a set* Λ *of sound models of the actions in A.*

According to the above definition, an action $a \in A$ of a domain is modelled as a function $\alpha \in \Lambda$ that maps models into models. Naturally, in order to be able to make the *process of reasoning about* (i.e., simulating) action fully automatic, one must specify a general algorithm Γ that calculates the model $\alpha(M)$ for any given action model $\alpha \in \Lambda$ and any world model $M \in \Sigma$. For this to be possible, all functions in Λ (and models in Σ) will have to be represented *declaratively*, so that their representation can be given as input to the procedure Γ.

2.1 The Analogical structures: SetGraphs

Having completed the formalisation of the basic framework for hybrid, model-based domain descriptions, let us move on to the description of the diagrammatic side of the representation:

Definition 10 – *A* nodeSet *is an abstract structure defined recursively as follows:*

- *a* nodeSet *is either a* '*node*' *or a* '*place*'*;*
- *a* node *is the empty-set element* $\emptyset$*;*
- *a* place *is a finite set of nodeSets.*

NodeSets are structures consisting of multi-nested sets of (empty) sets, with no limit on the level of nesting. For example, the structure { $\emptyset$, {{{$\emptyset$, $\emptyset$}}}, {{$\emptyset$}, { }, {$\emptyset$}} } is a nodeSet, where the syntax "{*x, y, …, z*}" denotes a place containing nodeSets *x, y, … z*. Notice the difference between a node (a *constantly* empty set '$\emptyset$'), and an empty place '{ }' (a nodeSet which just *happens* to be empty).

Definition 11 – *A* ground *nodeSet is a nodeSet in which every place is associated to a unique label and every node is associated either to an R-value (*$\in \Re_{\perp}$*) or to a unique label.*

For example, the nodeSet {{$\emptyset$},{ },{$\emptyset$}} could be ground to $P_2\{ P_1\{A\}, P_3\{ \}, P_4\{2.0\}\}$, where the syntax "*name*{ *x, y, … z* }" denotes a place with label "*name*" containing nodeSets *x, y, … z,* and the nodes are simply represented by the associated labels or numeric values.

Given a nodeSet N, $\wp(N)$ consists of the set of all the sub-nodeSets appearing in N. For example, if $N = P_2\{ P_1\{A\}, P_3\{ \}, P_4\{2.0\}\}$ then $\wp(N) = \{ P_1, P_2, P_3, P_4, A, 2.0\}$, where the labels and numeric values denote the corresponding nodeSets.

Definition 12 – *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 on* $\wp(N)$*,* $E_i \subseteq \wp(N)\times\wp(N)$*. All pairs* $(x,y) \in E_i$ *(or* 'edges'*) of relation* E_i *may be associated to a label (identical for all pairs), which will be the name of relation* E_i*.*

In order to represent setGraphs we shall adopt a graphical notation in which places are depicted as ovals and nodes as black dots. All (and only) the nodeSets that are contained in a place will be represented as elements appearing within the perimeter of the corresponding oval. All edges will be denoted as oriented arcs connecting two (not necessarily distinct)

elements of $\wp$(N). For example, if N= P_2{ P_1{A, B}, P_3{ }, P_4{5}} and E_1={(P_1, P_3), (P_3 , P_1), (P_2 , A), (A, 5)}, then Figure 1 contains a possible representation of setGraph ⟨N, {E_1}⟩.

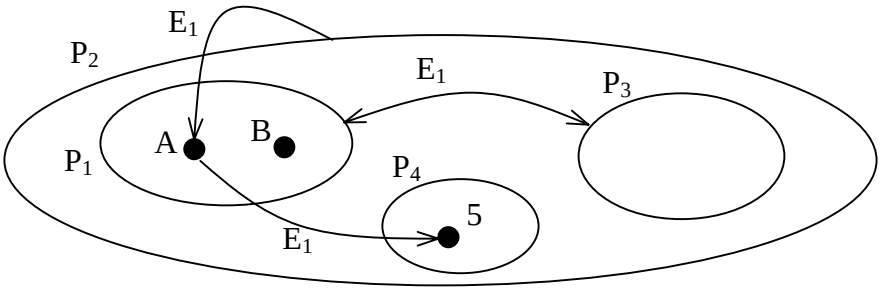

Figure 1. Example – Graphical representation of a setGraph.

Let us assume to have three (finite) trees (hierarchies) of labels having, respectively, "NODE", "PLACE" and "EDGE" as roots, and such that no identical labels appear in the same tree. All 'leaves' (terminal labels) of these hierarchies will be called *instances*; all nodes (non-terminal labels) *sorts*. Root labels are sorts. Every sort label *l* will represent the set of instances $\{l_1,\ldots,l_n\}$ of the sub-tree having '*l*' as root. Consider a set of variable names $\{v_1,\ldots,v_n,\ldots\}$ such that the type (domain) of a variable is either $\Re_\perp$ (*numeric* variable), or one of the sorts (*sort* variable). A *typed* setGraph is a setGraph augmented with sort labels and variable names:

Definition 13 – *Given three sort hierarchies NODE, PLACE and EDGE of unique labels and a finite set* $\{v_1, v_2,\ldots,v_k\}$ *of variable names, a* typed setGraph *is a setGraph in which each node, place and edge is associated either to (1) a label from the NODE, PLACE and EDGE hierarchy, respectively, or to (2) a unique sort-variable name having type set to one of the NODE, PLACE and EDGE sorts, respectively. A node may also be associated to a constant R-value* $x\in \Re_\perp$ *or to a unique numeric variable name.*

We assume that, for any given domain, the sort hierarchies H for nodes, places and edges are fixed and remain unchanged.Notice that in a typed setGraph a variable name must be unique, i.e., the variable may appear only once as a node, place or edge, whereas two nodes (or two places, or two edges) may bear the same sort label. We will refer to the label or variable name associated to a nodeSet as to the nodeSet's *identifier*.

Definition 14 – *An* instantiated *setGraph is a typed setGraph in which all elements are associated to either instances labels (terminal symbols, leaves), R-values, or numeric variables, and such that no two nodes or two places are associated to the same identifier.*

Definition 15 – *A* ground *setGraph is an instantiated setGraph containing no variables.*

Definition 16 – *Given a sort hierarchy H (consisting of NODE–PLACE–EDGE hierarchies) a typed setGraph* $T=\langle N,E\rangle$ *and a ground setGraph G, T is* satisfied *in G iff there exist a substitution* θ *of all sort variables in T with instances of corresponding sorts and a 1-1 function* σ:$T\rightarrow G$ *binding elements (nodeSets and edges) of T to elements of G such that:*

- *for all nodeSets* $x, y \in \wp(N)$, *if* $x\in y$ *then* $\sigma(x)\in\sigma(y)$;
- *for all edges* $e=(x,y)\in E$, $\sigma(e) = (\sigma(x), \sigma(y))$;

- *for each element x of T, if w_T and w_G are the identifiers or R-values of x and σ(x) after substitution θ, then either (1) $w_G = w_T$, or (2) w_G is an instance of w_T (i.e., $w_G \in w_T$), or (3) one is a numeric variable, and the other is an R-value or a numeric variable.*

For example, consider the typed setGraph depicted in Figure 2.(*a*). It is assumed that any element with unspecified label is taken to be associated to the root label of the corresponding hierarchy (e.g., edge (v_1,v_2) is associated to sort "EDGE"). The typed setGraph of Fig. 2.(*a*) is satisfied in the (ground) setGraph of Fig.1 (with sort hierarchy of Fig. 2.(*b*)) with substitution θ and binding σ, where θ =(v_1/A), and σ binds node labelled v_1 to node A, node v_2 to node with value 5, the outmost place (having label "PLACE") to P_2, the unlabelled place to P_1, place P_4 to P_4 (of Fig.1) and (v_1,v_2) to (A,5).

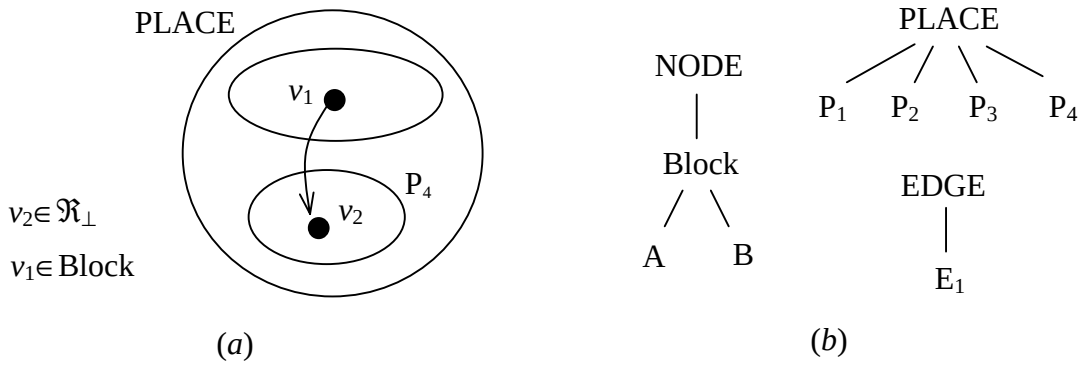

Figure 2. (*a*) A typed setGraph which is satisfied in the (ground) setGraph of Fig.1; (*b*) Sort hierarchy for the typed setGraph.

2.2 The Semantics of Planning with SetGraphs

Definitions 10-16 provide the tools necessary to construct domain models using setGraphs. In particular, given a domain with language $\mathcal{L} = \langle P, F, C\rangle$, a world model will consist of a pair (*d*,ε) (see Definition 4), where *d* will be a *ground* setGraph, and ε a 1-1 mapping from C to instance (leaf) labels of the NODE-PLACE-EDGE hierarchies.

Given a model (*d*,ε) and *n* terms $t_1,\ldots t_n \in C$, the mapping ε allows us to identify which elements in *d* symbols $t_1,\ldots t_n$ identify (see also Definitions 4-5). We can assume without loss of generality that the set of symbols C is a subset of the set of instance labels. In particular, for any model (*d*,ε), we will assume ε(*c*)= *c*. This enables us to 'forget' about the mapping ε: in what follows, a model (*d*,ε) is represented simply by a ground setGraph *d*.

According to Definition 9, in order to represent a given domain *Dom*=⟨S,A⟩ one must provide a domain representation structure $\mathcal{R} = \langle \Psi, \Phi, \Delta\rangle$ for $\mathcal{L}$, a set Σ of models in $\mathcal{R}$ that represents S, and a set Λ of (sound) models of the actions in A. Each of these elements, of course, will be domain-specific. The set Δ will be the set of ground setGraphs that appear in the set of models Σ. The procedures in Ψ (and Φ) for logical symbols will ascertain the presence of specific elements (and return a specific R-value contained) in a ground setGraph – the syntax for their specification (see below) relies upon Definition 16. The other procedures in Ψ and Φ will calculate standard relations (such as '=','>', etc.) and operations (including '+', '–','·', '/', etc.) on ℜ. Given the set of possible models Σ (ground setGraphs) representing S, an action model representation O will specify a transformation of a model $d \in \Sigma$ into another model $d'=O(d) \in \Sigma$. The syntax for the specification of the action

representations and of procedures Ψ, Φ may vary according to the needs. A relatively simple syntax is proposed below. More complex and expressive formalisms for describing setGraph inspections and transformations can also be defined.

Let us begin with the specification of the syntax and semantics for the procedures in Ψ and Φ that are associated to logical relation and function symbols of P and F, respectively:

- *A procedure $\psi_p \in \Psi$ associated to an n-placed logical numeric relation symbol $p \in P$ is described by a pair (b, T_p), where $b \in \{True, False\}$ and T_p is a typed setGraph containing (exactly) n sort-variable names $v_1,\ldots,v_n$ such that $v_1,\ldots,v_n$ are of the sorts $D_p^1,\ldots, D_p^n$ specified for the arguments of $p \in P$ (i.e., $\forall i \in \{1,\ldots n\}, v_i \in D_p^i$);*
- *A procedure $\phi_f \in \Phi$ associated to an m-placed logical function symbol $f \in F$ is described by a pair (T_f, w), where T_f is a typed setGraph containing (exactly) m sort-variable names $v_1,\ldots, v_m$ (such that $v_i \in D_f^i$) and one numeric-variable name w (associated to a node).*

Definition 17 – *Given a language $\mathcal{L} = \langle P,F,C \rangle$ and the sort hierarchies H of a domain, let (b, T_p) be a pair describing a procedure $\psi_p \in \Psi$, and (T_f, w) be a pair describing a procedure $\phi_f \in \Phi$ (assume that both T_f and T_p contain sort variables $v_1, \ldots, v_n$). Given a model d, and a logical atom $p(t_1,\ldots,t_n)$ (PNE $f(t_1,\ldots,t_n)$) of $\mathcal{L}$, let T_p' (T_f') be the setGraph obtained by replacing $(v_1/t_1,\ldots,v_n/t_n)$ in T_p (T_f). The value returned by procedure ψ_p (ϕ_f) is defined as:*

- $\psi_p(d, t_1,\ldots,t_n) = \neg (b \oplus x)$, *where x=True iff (typed) setGraph T_p' is satisfiable in d*
- $\phi_f(d, t_1,\ldots,t_n) = k \in \Re_\perp$ *if, for each binding σ such that setGraph T_f' is satisfied in d with σ, $\sigma(w) \in d$ is always the same node, having value k;* $\phi_f(d, t_1,\ldots,t_n) = \perp$ *otherwise*

The result of $\neg (b \oplus x)$ (negated x-or) is equal to *x* if *b* is *True*, $\neg x$ otherwise. Hence, setting *b=True* (*False*) means that ψ_p will return *True* iff the setGraph T_p' *is* (is *not*) satisfiable in *d*.

Having defined the syntax and semantics of the procedures in Ψ and Φ of the domain representation structure $\mathcal{R}$, let us move on to the modelling of action. In what follows, we assume that the language $\mathcal{L} = \langle P, F, C \rangle$, sort hierarchies H and representation structure $\mathcal{R}$ have been determined and remain constant for the specific domain considered $Dom=\langle S,A \rangle$.

A (sound) model α of an action $a \in A$ is a transformation $\alpha:\Sigma \rightarrow \Sigma$ of ground setGraphs into ground setGraphs. An action model α is described by an *instantiated action schema* (or *step*) $O = (\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$, where:

- *Π_B is an instantiated setGraph (possibly containing numeric variable-names);*
- *Ω is a finite list of typed setGraphs (possibly containing variables names of Π_B);*
- *L is a (finite) set of numeric atoms of $\mathcal{L}$, some of which may be negated;*
- *Π_A is an instantiated setGraph (possibly containing variable names appearing in Π_B);*
- *V is a (possibly empty) set of assignment operations in the form "w_j Op expr", where w_j is a numeric variable name that appears in Π_B, 'Op' is an element of the set {=, +=, *=, –=, /= }, and 'expr' is either a PNE or NE of $\mathcal{L}$, or a variable appearing in Π_B.*

The instantiated setGraph Π_B (preconditions) describes the situation holding in the model *before* the application of the action, while setGraph Π_A (effects) describes the situation *after*. The setGraphs in Ω (negative preconditions) describe situations that must *not* hold in a model in order for O to be applicable in it. Set L specifies a set of numeric conditions (possibly

containing PNEs and NEs) that must be satisfied. V contains a set of update operations for computing new values of some of the nodes (identified in Π_B by numeric variables) of the current model. Intuitively, an instantiated action-schema should be applicable in a model *d* (representing state *s*) whenever Π_B is satisfied in *d*, none of the setGraphs in Ω is satisfiable in *d*, and all numeric atoms of L are satisfied in *s*:

Definition 18 – *Given a world model (ground setGraph) d, an* instantiated action schema *(or* step*) $O=(\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ is* applicable in *d* with binding σ *iff the setGraph Π_B is satisfied in d (with mapping σ), not one of the setGraphs in Ω is satisfiable in d, and, for each positive (negative) numeric atom $q(t_1, \ldots t_n)$ ($\neg\, q(t_1, \ldots t_n)$) of L, $\psi_q(e(t_1), \ldots, e(t_n)) =$ True (False), where $\psi_q \in \Psi$ is the procedure associated to the n-placed numeric relational symbol $q \in P$, and e( ) is the evaluation function defined in Proposition 1.*

Thanks to Definition 5 and Proposition 1, if *d* represents a state *s* then for O to be applicable in *d* all positive (negative) atoms of L must be satisfied (not satisfiable) in *s*.

Given a step $O=(\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$, the setGraph Π_A represents the arrangement of all elements of Π_B after the execution of O. The possible transformations considered here are: (*a*) *addition* or *removal* of an element, (*b*) *movement* of a nodeSet, and (*c*) re-assignment (or update) of an R-value associated to one of the nodes. The movement and removal of nodeSets in a setGraph is based on the following 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.*

Movement and removal of elements in Π_B are encoded implicitly by setGraph Π_A. The different nodeSets of Π_B and their (possibly new) positions are identified in Π_A using the same identifiers that those elements have in Π_B. Since addition of elements is permitted, Π_A might also contain new nodeSets (associated to labels or values that do not appear in Π_B) or new edges. Similarly, since removal is permitted, Π_B might contain nodeSets or edges that do not appear in Π_A. Update effects (*c*) are described by set V, where the update of the R-value $x \in \Re_\perp$ of a node identified by variable w_j in Π_B is described by an expression "w_j *Op expr*" $\in$ V, where the operators '=', '+ =', '– =', '* =' and '/ =' correspond, respectively, to `assign`, `increase`, `decrease`, `scale-up` and `scale-down` effects in PDDL2.1 (Fox & Long 03)). Notice that all elements of Π_B not moved, removed or updated are left unaltered (i.e., we assume *default persistence).*

Let us now describe how to obtain the new (ground) setGraph *d'*=O(*d*), resulting from the application of step O to model *d*. Let $O=(\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ be a step applicable in *d* with binding σ (if O is not applicable in *d*, we assume O(*d*)= *d*). The transformation of Π_B defined by O unambiguously identifies, through the binding σ, a corresponding transformation of *d*, as each element $x \in \Pi_B$ is bound by σ to a (distinct) element $\sigma(x) \in d$. Any transformation of an element $x \in \Pi_B$ will be interpreted as a transformation of the corresponding element $\sigma(x)$:

Definition 19 – *Given an instantiated action schema $O=(\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ applicable in model d with binding σ, the* result *of applying O to d is a new setGraph d'=O(d) obtained by applying the following transformations of elements of Π_B to the corresponding elements of d:*

1. *Each (ground) nodeSet x of Π_A not appearing in Π_B and contained in a place X of Π_A is added to Π_B as a new element of place X (if $X \notin \Pi_B$, X should be added first);*
2. *For each edge (x,y) of Π_A not appearing in Π_B, a new edge (associated to the label of (x,y) in Π_A) is added to Π_B , linking the corresponding nodeSets (x,y) in Π_B;*

3. *Each element (nodeSet or edge) of Π_B not present in Π_A is removed from Π_B (if a nodeSet is removed, all edges linked to it and all its contents are removed as well);*
4. *Each nodeSet of Π_B that appears in a different place in Π_A is moved from its current place to the new one (all the contents of and edges linked to a nodeSet move with it);*
5. *For each assignment operation "w_j Op expr"$\in V$, an R-value $x_j' = (x_j \; Op \; y) \in \Re_\perp$ is calculated and stored, where $x_j \in \Re_\perp$ is the current R-value (in Π_B) of node w_j, (x_j Op y) is the value assigned to x_j by the 'C' operation 'x_j Op y' ($\perp$ if ($x_j = \perp$) or ($y = \perp$)), and $y = e(expr)$, where the function 'e( )' is defined as in Proposition 1 (and $e(w_k) = x_k$);*
6. *For each operation "w_j Op expr"$\in V$, the new R-value of node w_j in Π_B is set to x_j'.*

Notice that the order in which the operations in V are considered is irrelevant, as expected.

In order to allow the specification of more general action schemata, instance labels of a step may be replaced with appropriate sort labels or with variable names of the corresponding sort. This produces a *parameterised* action schema $O(v_1,\ldots, v_n)$ with parameters $v_1,\ldots, v_n$ of specific sorts. The action schema $O(v_1,\ldots, v_n)$ represents the (finite) set of *instantiated* steps that can be obtained by replacing (everywhere in O) all variables and sort labels appearing in Π_B with instances of that sort (notice that $v_1,\ldots, v_n$ do not contain numeric variables).

The last definition required concerns the description of goals using setGraphs, and the condition required for a goal to be satisfied in a world model (ground setGraph). An '*atomic goal expression*' G is defined as a triple (Π, Ω, L), analogous to the precondition of a (parameterised) setGraph action-schema: Π is a typed setGraph, Ω is a list of typed setGraphs (containing only variables of Π), and L is a set of (possibly negative) numeric atoms of $\mathcal{L}$.

Definition 20 – *An atomic goal expression $G=(\Pi, \Omega, L)$ is satisfied in a model d iff $\exists$ a substitution θ of all sort variables of Π with instances of the corresponding sorts such that $\Pi(\theta)$ is satisfied in d, none of the setGraphs in $\Omega(\theta)$ is satisfiable in d, and for each positive (negative) numeric atom $q(t_1, \ldots t_n)$ ($\neg q(t_1, \ldots t_n)$) of L, $\psi_q(e(t_1), \ldots, e(t_n)) = True$ (False).*

Example 2 – Consider the BW domain BW=⟨S,A⟩ of Example 1, with associated language $\mathcal{L}_1$ extended with the 2-placed logical relation symbol "On". S and A are the usual sets of possible states and actions of BW, except that the blocks have a (fixed) weight. Let us specify a domain representation structure $\mathcal{R}_2 = \langle \Psi_2, \Phi_2, \Delta_2 \rangle$ for $\mathcal{L}_1$ using setGraphs. The NODE hierarchy for this domain contains the sort *Object* having two sub-sorts, *Block*={B_1, B_2, B_3} and *Table*={T}. The PLACE hierarchy contains ten instances, $P_1,\ldots,P_{10}$, and the EDGE hierarchy contains three instances, "Above", "On" and "Weight". Figure 3 illustrates a ground setGraph representing a BW state in which all blocks are lying on the table. The edge labels (omitted to avoid cluttering) are identified by three distinct types of arrow: dashed (denoting label "Weight"), bold (denoting "On") and normal arrows (denoting "Above").

The set of models Δ_2 consists of the set of all possible ground setGraphs that can be obtained from that of Fig. 3 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 *x*, *y* connected by an "On" edge (*x*,*y*) such that *x* contains a node and *y* does not). The procedures $\psi_{Above}(d,v_1,v_2)$, $\psi_{On}(d,v_1,v_2)$ and $\phi_{Weight}(d,v_1)$ are represented by the pairs (*True*,T_{Above}), (*True*,T_{On}) and (T_{Weight}, w_1), respectively (see Figures 4.(*a*), 4.(b) and 4.(*c*), respectively). Procedures $\psi_{\geq}$, ϕ_+, ϕ_-, ϕ_* and $\phi_/$ are given their standard interpretation (as in Example 1).

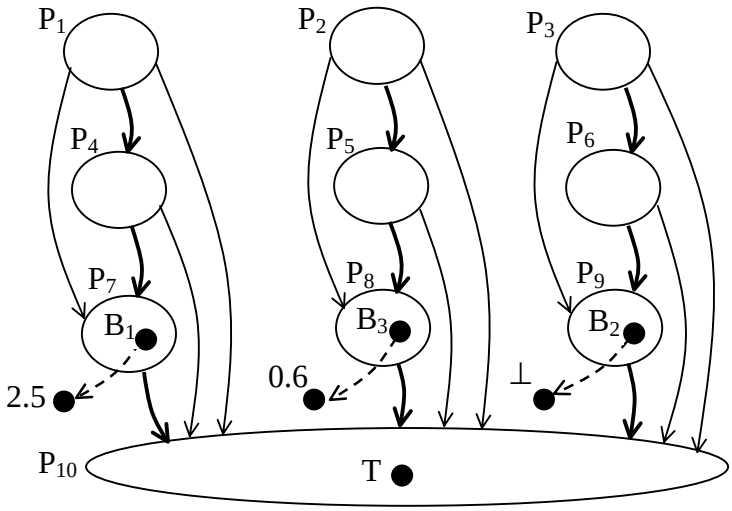

Figure 3. A model (ground setGraph) representing a Blocks-World state with weighted blocks.

The description of the (parameterised) action schema Move(x,y,v_1,v_2)=(Π_B, Ω, L $\Rightarrow$ Π_A,V), representing the (sound) model of the action consisting of picking up a (clear) block x (from place v_1) and putting it on object y (in place v_2) is depicted in Figure 5 (notice that L=V= $\varnothing$).

As usual, any unlabelled place in Figures 4 and 5 is assumed to be associated to the corresponding sort-hierarchy root label, "PLACE". The setGraphs of Figures 5.(Ω_1) and 5.(Ω_2) are elements of Ω, and describe the negative preconditions of the action schema: (Ω_1) requires that there exists no place on the place containing x such that it contains a block (i.e., block x is clear); (Ω_2) requires that there exists no block inside place v_2 (i.e., v_2 is empty).

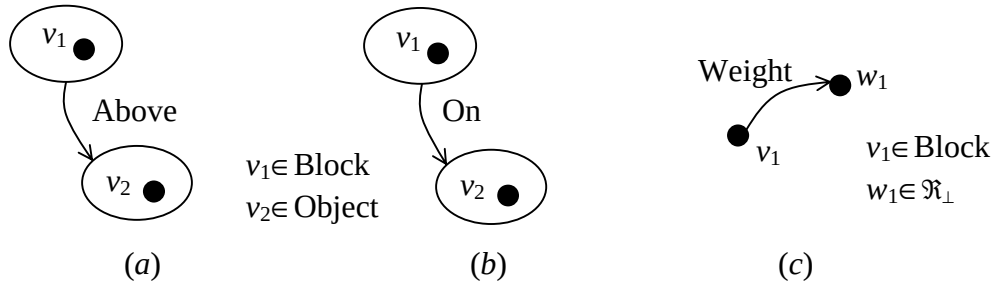

Figure 4. Typed setGraphs: (*a*) T_{Above}, (*b*) T_{On}, and (*c*) T_{Weight}, representing ψ_{Above}, ψ_{On} and ϕ_{Weight}.

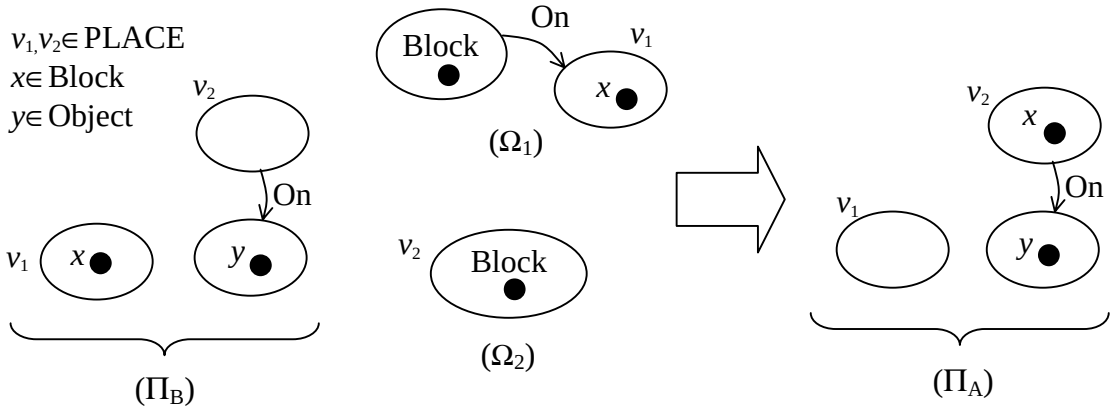

Figure 5. The action schema Move(x,y,v_1,v_2). (Π_B): precondition setGraph; [(Ω_1),(Ω_2)]: negative precondition list Ω; (Π_A): effect setGraph.

2.3 Hybrid Planning Representations

Assume that the action 'Move' of the previous Example is subject to the requirement that the weight of the block being moved be less than a certain real value, e.g., 2. There are two equivalent ways in which this constraint could be added to the action schema. The first one consists of simply adding the condition "≥(2, Weight(*x*))" to the set L. The second one is to modify the precondition Π_B by adding a new edge with label "Weight" that links *x* to a new node associated to a numeric variable name (e.g., *wgt*), and add the condition "≥ (2, *wgt*)" to L. The former solution relies on the *interpretation* of the representation (T_{Weight}, w_1) (see Fig. 4.(*b*)), which can be used in order to check for the truth value of the PNE 'Weight(*x*)' (as specified by the semantics of procedure ϕ_{Weight}). The latter *integrates* the representation of procedure $\phi_{Weight}(d, v_1)$ in the action schema itself. Namely, the typed setGraph $T_{Weight}(v_1,w_1)$ is added to the precondition setGraph (after its variables have been replaced by '*x*' and '*wgt*', respectively), and the label replacing variable w_1 is used in L instead of the PNE 'Weight(*x*)'.

This example underlines that, if all of the procedures ϕ_{fj} associated to the function symbols f_j are represented declaratively using setGraphs, the propositional formulation of a PNE $f_j(t_1,\ldots,t_m)$ and its equivalent setGraph representation can be used interchangeably in the description of an action schema. Provided that each logical relation symbol $p \in P$ also has a corresponding procedure ψ_p described declaratively using setGraphs, the set L of the action schema could be allowed to include also any *logical* atom of $\mathcal{L}$. For example, adding an instance of the logical relation "Above(v_1,v_2)" to the set L of an action schema would have the same effect of adding the equivalent setGraph of Figure 4.(*a*) to the precondition Π_B.

Notice that some of the logical atoms of $\mathcal{L}$ may be represented by a "negative" setGraph (and that a negative logical atom of $\mathcal{L}$ may be represented through a "positive" setGraph). For example, consider the 1–placed logical relation symbol "Clear" for the BW domain, having argument of sort 'Block'. The procedure ψ_{Clear} could be represented by the pair (*False*, T_{Clear}), where $T_{Clear}(x)$ is the typed setGraph of Figure 5.(b_1). In order to require the (positive) precondition "Clear(*x*)", the setGraph $T_{Clear}(x)$ should be added to the (negative) precondition list Ω, rather than to Π_B. On the other hand, in order to require the negative atom "not Clear(*x*)", the setGraph $T_{Clear}(x)$ should be added to the (positive) precondition setGraph Π_B.

Consider a propositional domain representation structure $\mathcal{P}$ (for a language $\mathcal{L}$) analogous to the one adopted for the semantics of PDDL2.1, level 2 (Fox & Long 03), in which a model M consists of a set of (logical) atoms a (finite) array of R-values, representing the values assumed by each of the PNEs in model M (see also Example 1). Given a (sound) propositional action schema for transforming (propositional) models, is it possible to build an *equivalent* setGraph description? As demonstrated below, the answer to this question is "*yes*". The proof of this property follows directly from the two lemmata provided below:

Lemma 1 – *Given a finite set C containing k constant symbols* $\{c_1,\ldots, c_k\}$*, any n-ary relation* $p \subseteq C^n$ *on C can be represented as a setGraph.*

Proof – Consider a setGraph R containing *k* nodes having unique labels $\{c_1,\ldots, c_k\}$= C and two places labelled "*True*" and "*False*". For each of the possible k^n tuples $p_i = (c_{i1},\ldots,c_{in}) \in C^n$, R will contain a node labelled "p_i" and *n* edges $(p_i,c_{i1}),\ldots,(p_i,c_{in})$, labelled $\alpha_1,\ldots,\alpha_n$. For all tuples $p_i \in p$, node p_i will be inside place "*True*"; for all the remaining tuples $p_j \notin p$, node p_j will be inside "*False*". Given $p_x = (c_{x1},\ldots,c_{xn}) \in C^n$, $p_x \in p$ *iff* the typed setGraph T_{p_x}=(N,E) (with node hierarchy containing sort "π_p"=$\{p_1,\ldots, p_{kn}\}$) consisting of nodeSet N={*True*$\{\pi_p\}$, $c_{x1},..,c_{xn}\}$ and set of labelled edges E=$\{\alpha_1(\pi_p, c_{x1}), \ldots,\alpha_n(\pi_p, c_{xn})\}$ is satisfied in R.

Lemma 2 – *Given a finite set C containing k constant symbols* $\{c_1,\ldots, c_k\}$*, any function* $f:C^m\rightarrow\Re$ *of m arguments in C can be represented as a setGraph.*

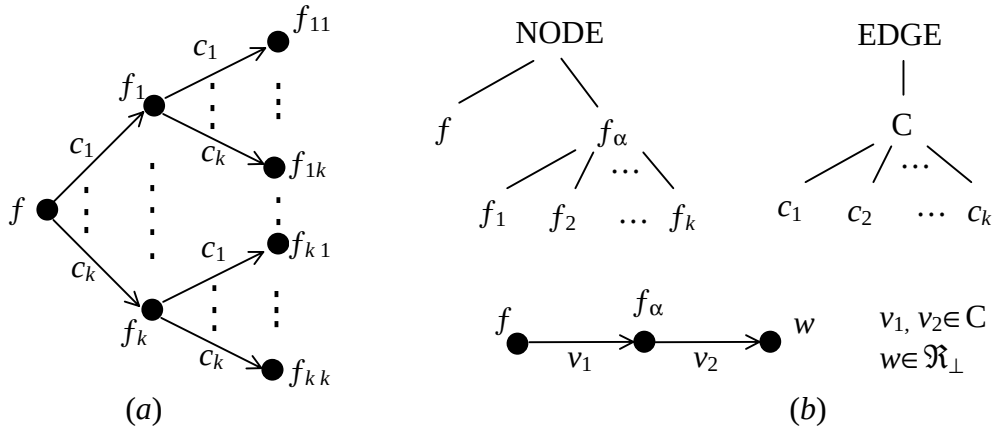

Figure 6. (*a*) ground setGraph encoding function *f* of two arguments; (*b*) typed setGraph $T_f(v_1,v_2)$.

The proof of Lemma 2 is trivial, and is not included. However, Figure 6.(*a*) shows how to encode a function with two arguments $f:C^2\rightarrow\Re$, where $f_{i\,j}$ is the R-value $f(c_i,c_j)$ ($f_{i\,j}=\perp$ if $f(c_i,c_j)$ is undefined), for $i,j\in\{1,\ldots, k\}$. Figure 6.(*b*) contains the typed setGraph T_f (and associated NODE and EDGE hierarchies) for representing procedure $\phi_f(d, v_1, v_2)\equiv(T_f, w)$. By induction, it is easy to see that any function $f:C^m\rightarrow\Re$ with *m* arguments in C can be encoded using analogous setGraphs, in which the levels of edges are expanded to *m*. Notice that setGraph T_f has 'matching' time linear in *m*.

Lemma 1 and 2 show how any propositional model M can be encoded as an equivalent (ground) setGraph model. This result should not come as a surprise, since it is well known that semantic networks (e.g., Sowa's Conceptual Graphs (Sowa 84)) are expressively equivalent to first-order logic, and it is not difficult to see that setGraphs can encode such structures. However, the important point to note here is that in many cases a setGraph representation allows several of the relevant relations of a domain to be encoded 'implicitly' in the model. This simplifies the representation, avoiding the problem of ramification of effects, and significantly increasing planning efficiency (see Section 2.4).

Let us now show how, given a propositional action schema, it is possible to build an *equivalent* setGraph description using the domain representation structure described above.

Consider ground propositional action schemata of form $\Gamma\Rightarrow$ E, where Γ is a list of preconditions (a set of possibly negative logical and numerical atoms of $\mathcal{L}$) and E is a list of effects, including logical (possibly negative logical atoms) and numeric effects (a set of update operations "*Op*(PNE, *expr*)", where *Op* is any of "`assign`", "`increase`", "`decrease`", "`scale-up`" and "`scale-down`", and '*expr'* is any NE or PNE of $\mathcal{L}$).

Definition 21 – *Given a domain* $\langle S,A\rangle$ *with language* $\mathcal{L}$*, let* $\varphi_a=(\Gamma\Rightarrow E)$ *be a sound, ground propositional description of an action* $a\in A$*, and let* $\mathcal{R}=\langle\Psi,\Phi,\Delta\rangle$ *be a setGraph-based domain representation structure for* $\mathcal{L}$ *such that every logical relation* $p\in P$ *of* $\mathcal{L}$ *(consisting of a set of logical atoms* $p(t_1,\ldots t_n)$*) is represented as specified in the proof of Lemma 1, and every logical function* $f\in F$ of $\mathcal{L}$ *(consisting of a set of PNE* $f(t_1,\ldots t_m)$*) is represented as in*

Figure 6.(a). An instantiated setGraph action-schema γ_a equivalent *to* φ_a *is built by progressively adding elements to* $(\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ *as explained below (where, initially, both* Π_B *and* Π_A *contain only nodes* $\{c_1,\ldots c_k\} = C$ *and two empty places "True" and "False", and* Ω, L *and V are empty):*

- *For each logical atom* $p_x = p(t_1,\ldots t_n)$ *of* $\mathcal{L}$, $x \in \{1,\ldots,k^n\}$, *let* η_x *be a node with label "*p_x*" (an instance of sort* π_p*) and let* λ_x *be the set of labelled edges* $\{\alpha_1(p_x, t_1),\ldots, \alpha_n(p_x, t_n)\}$

1. *For each positive logical atom* $p_i \in \Gamma$, *add node* η_i *to "True" in* Π_B, *and add set* λ_i *to* Π_B
2. *For each negative logical atom* $\neg p_j \in \Gamma$, *add node* η_j *to place "False" in* Π_B, *and* λ_j *to* Π_B
3. *For each positive logical atom* $p_i \in E$, *add node* η_i *to place "True" in* Π_A *and add* λ_i *to* Π_A *(if* $\eta_i \notin \Pi_B$, *add* η_i *to place "False" in* Π_B, *and add set of edges* λ_i *to* Π_B*)*
4. *for each negative logical atom* $\neg p_j \in E$, *add node* η_j *to place "False" in* Π_A, *and* λ_j *to* Π_A (*if* $p_j \notin \Gamma$, *then also add node* η_j *to place "True" in* Π_B, *and* λ_j *to* Π_B)
5. *for each numerical update Op*(PNE, *expr*)$\in E$, *where* PNE $= f_j(t_1,\ldots t_m)$, *add the setGraph representation* $T_{f_j}(t_1,\ldots t_m)$ *(Fig. 6.(b)) of* $f_j(t_1,\ldots t_m)$ *to both* Π_B *and* Π_A, *and add the update operation "*w_j *Op expr" to V (*w_j *is a numeric variable-name identifying PNE* $f_j(t_1,\ldots t_m)$*)*
6. *add each (possibly negative) numeric atom q (*$\neg q$*) of* Γ *to L (all occurrences of PNEs* $f_j(t_1,\ldots t_m)$ *may be replaced with the corresponding numeric variable '*w_j*' used in step* 5*).*

Notice that it is assumed here that all the negative atoms of E ('deleted' by φ_a) are satisfied in any model M to which φ_a is applicable, and that all the positive atoms of E ('added' by φ_a) are *not* satisfiable in any of such models.

Theorem 1 – *Given a language* $\mathcal{L}$ *for a domain* $\langle S,A\rangle$, *a propositional domain representation structure* $\mathcal{P}$ *for* $\mathcal{L}$, *a sound, ground propositional action-schema* φ_a *encoding action* $a \in A$, *and a setGraph-based domain representation structure* $\mathcal{R} = \langle \Psi, \Phi, \Delta\rangle$ *for* $\mathcal{L}$ *using representations specified in Lemma 1 and Fig.6(a), the (instantiated) setGraph action-schema* $\gamma_a = (\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ *equivalent to* φ_a *is such that for any propositional model M and state* $s \in S$ *such that* $M \cong_{\mathcal{P}} s$ *and* $\forall d \in \Delta$, *if* $d \cong_{\mathcal{R}} s$ *and* $\varphi_a(M) \cong_{\mathcal{P}} s'$, *then* $\gamma_a(d) \cong_{\mathcal{R}} s'$.

The proof follows directly from (Definition 19), from Lemma 1 and 2, and from the way in which the equivalent action-schema γ_a is built from φ_a (Definition 21).

Example 3 – Consider the language $\mathcal{L}_2 = \langle P_2,F_2,C_2\rangle = \langle\{On,Clear, \geq\}, \{Weight, Fuel,+,–,*,/\}, \{T, B_1, B_2, B_3\}\rangle$ for a BW domain with 'weighted' blocks, where 'Fuel' is a 0-placed function returning the total amount of energy available to the robot. Assume that the propositional action schema $\varphi_{Move}(x,y,z) = (\Gamma \Rightarrow E)$, encoding the "Move" action, is as follows:

$\Gamma = \{$ `On(`x`,`y`)`, `Clear(`x`)`, `Clear(`z`)`, $\geq$`(2.0,Weight(`x`))`, $\geq$`(Fuel,5.0)`$\}$
$E = \{$ `On(`x`,`z`)`, `Clear(`y`)`, $\neg$`On(`x`,`y`)`, $\neg$`Clear(`z`)`, `decrease(Fuel,3.5)`$\}$

where $x, y, z \in$ *Block*$=\{B_1, B_2, B_3\}$. The action schema $\gamma_{Move}(x,y,z, v_1,\ldots v_5) = (\Pi_B, \Omega, L \Rightarrow \Pi_A, V)$ equivalent to $\varphi_{Move}(x,y,z)$ (built using Definition 21) is depicted in Figure 7 (with $\Omega = \varnothing$, L$=\{$ $\geq$`(2.0,`w_1`)`, $\geq$`(`w_2`, 5.0)`$\}$, V$=\{w_2 - = 3.5\}$). SetGraph Π_A is identical to Π_B except that nodes v_4, v_2 have migrated from *True* to *False*, and nodes v_3, v_5 have moved the opposite way.

Notice that $\gamma_{Move}(x,y,z)$ is actually a parameterised action-schema, obtained by a generalisation of the result of Definition 21. As described in the proof of Lemma 1, π_{Clear} and

π_{On} are node sorts such that π_{Clear}={$Clear_1$,$Clear_2$,$Clear_3$} and π_{On}={On_1,…,On_9}, where each instance corresponds to one of the logical atoms of $\mathcal{L}_2$ built using symbols "Clear" and "On".

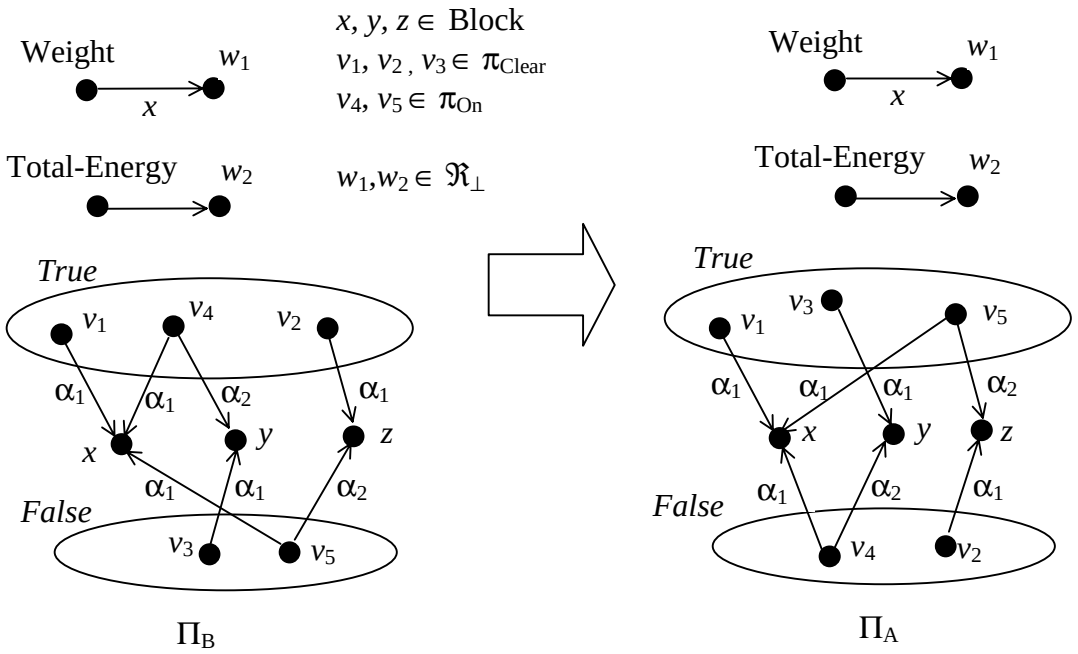

Figure 7. SetGraph action-schema $\gamma_{Move}(x,y,z)$ equivalent to propositional action-schema $\varphi_{Move}(x,y,z)$.

Although Definition 21 describes a universal procedure for producing a setGraph encoding equivalent to any given propositional action-schema, other, more convenient equivalent setGraph representation of a domain may exist, in which some of the relevant relations are encoded *implicitly* so as to simplify the representation and improve planning efficiency. For example, Figure 8 depicts a different version of the ("logical" part) of the action schema Move(x,y,z) (the numeric part is identical to that of Figure 7, and has been omitted). The representation of the procedure ψ_{On} for relation "On" is the same as in Example 2 (see Figure 4.(*b*)), while ψ_{Clear} is represented by the pair (*True*,T_{Clear}) (Figure 8.(*c*)), and the node "Clear" is a constant element of the current state.

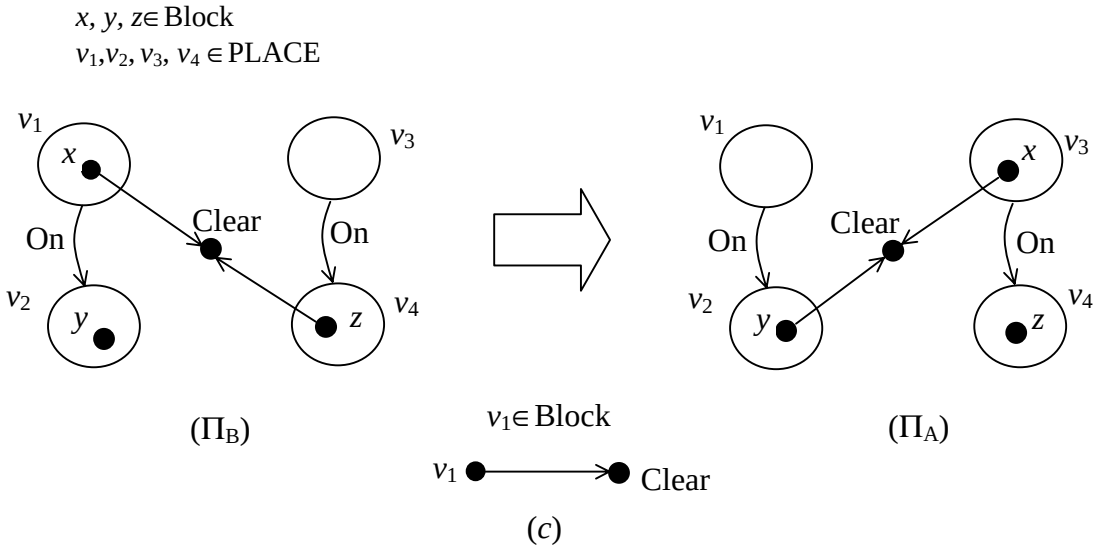

Figure 8. ($\Pi_B \Rightarrow \Pi_A$): simpler setGraph action-schema $\gamma_{Move}(x,y,z)$, equivalent to $\varphi_{Move}(x,y,z)$; (*c*) SetGraph T_{Clear} , representing procedure $\psi_{Clear}(d,v_1)$.

In view of Theorem 1 and Definition 21, it is easy to show that any PDDL2.1, level 2 (i.e., no durative actions) domain description can be compiled into an equivalent setGraph representation. In fact, all conditional effects and quantified formulæ of an action schema can be compiled away, obtaining an equivalent set of ground operators with disjunctive preconditions (see (Fox and Long 2003)). These, too, can be easily eliminated from the action schemata using pre-processing techniques (e.g., see (Gazen and Knoblock 1997)). Hence, any propositional action schema can be instantiated into a ground step consisting of preconditions (a conjunction of possibly negative logical and numeric atoms) and effects, i.e., a conjunction of (possibly negative) logical atoms and a conjunction of numerical effects (consisting of assignment operations on PNEs as specified in PDDL2.1). However, Definition 21 shows how the result can be transformed into an equivalent setGraph action schema. Finally, in order to be able to encode disjunctive goals, it is sufficient to allow atomic goals to be combined into disjunctive expressions: a disjunctive goal will be satisfied in a model *d* iff any of its atomic goals is satisfied in *d*.

2.4 PMP: A Prototype of purely analogical planner

The theoretical framework presented in the previous sections allows the specification of a *continuum* of domain descriptions, in which different models can have different portions of the language $\mathcal{L}$ interpreted in terms of setGraphs (we assume that all *numeric* symbols of $\mathcal{L}$ are always given the standard interpretation in terms of relations and operations on $\Re$). At one extreme of the continuum, no mapping between the logical relation and logical function symbols of $\mathcal{L}$ and setGraphs is provided: in this case, the representation is purely *analogical*, and sets L and V of an action schema cannot make use of any PNE expressed in propositional form. At the other extreme, each logical atom and each PNE of $\mathcal{L}$ has been assigned an equivalent (declarative) procedure in ψ and ϕ: in this case, the representation is fully *hybrid*, as any propositional action schema built from $\mathcal{L}$ can be replaced with the equivalent setGraph description. In between these two extremes lie various levels of hybrid descriptions, in which only some of the symbols of $\mathcal{L}$ have a setGraph interpretation and can be replaced with the equivalent analogical representation.

To exemplify, let us go back to Example 2. Suppose that the set of models Δ_2 (see Figure 3) and the Move(x,y,v_1,v_2) action-schema (Figure 5) were given, but the mappings between logical symbols of $\mathcal{L}$ and corresponding procedures (Fig. 4) were unknown. Then, the domain representation would allow the correct solution of BW problems, but the sets L and V of the action schema could not be augmented with any propositional expression like "Above(x,y)", "On(x,y)" or "Weight(x)". The gradual introduction of the mappings of Figure 4.(*a*), 4.(*b*) and 4.(*c*) would allow higher and higher levels of hybrid integration, until the propositional action schema $\varphi_{Move}(x,y,z)$ of Example 3 could be fully replaced with the equivalent setGraph action schema Move(x,y,v_1,v_2) of Figure 8 (assuming ψ_{Clear} is defined as in Figure 8.(*c*)). (Notice that, contrary to what required by Definition 21, in Figure 8 the relations and functions of $\mathcal{L}$ are not represented as explained in the proofs of Lemma 1 and 2. Hence, a more general version of the procedure given in Definition 21 should be devised in order to obtain this schema *automatically* from the description of the propositional action-schema φ_{Move}).

An example of syntax of a simple, purely analogical, domain description language based on the proposed framework is illustrated in (Garagnani & Ding 2003), together with the experimental results obtained with a prototype planner implementing such formalism. The description language developed represents a simplified version (a "Cartesian" restriction) of the general setGraph formalism. In particular, in the implementation, places are only allowed

to have an internal structure consisting of a matrix (or vector) of adjacent sub-places. In addition, edges are not permitted, although places and nodes can be assigned the same label to encode 'links' between elements. In short, a place can be either a *set* of nodes (represented as strings), or a one- or two-dimensional *array* of cells, each one allowed to contain up to one node (string). In addition, action schema cannot contain negative preconditions. In spite of the limited expressiveness, this language still contains many of the important characteristics of the general model, and allows an assessment of its effectiveness and performance against purely propositional systems. As an example of domain description, Figure 9 contains the encoding of the BW domain and of the Sussman's anomaly problem instance. The interpretation of this description is quite intuitive and left to the reader – the formal specification of the language is given in (Garagnani & Ding 2003).

Other five planning domains taken from the literature (including Eight-puzzle, Miconic, Briefcase, Gripper and Homeowner) were translated into equivalent analogical representations, and several planning instances were solved using the analogical planner (called 'Pattern Matching Planner', or PMP). A second, purely propositional planner (called 'PP') identical to PMP in the search algorithm (breadth-first forward state-space search) was also implemented and used to solve the same set of problems in propositional form. It should be noted that for all of the domains used, each of the problem instances was described so as to present exactly the same *search space* in both of the representations. The results obtained showed a clearly superior performance of PMP on all of the domains, and on all of the problem instances. The speed-up factor varied from a minimum of two to as much as ninety times faster. The speed-up demonstrated derives from the ability of the analogical representation to (*i*) capture *implicitly* the basic physical and topological *constraints* of the domain, and hence perform efficient updates of the current state avoiding the frame and ramification problem; and (*ii*) *organise* in appropriate data structures the entities of the domain, allowing efficient look-up operations.

```
(define (domain Blocksworld)
   (:ObjectTypes Block Table)
   (: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 9. Blocks World domain description for the PMP analogical planner, and Sussman anomaly.

3 Related Work and Discussion

Research in AI and knowledge representation has since long 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) (Koedinger 1992)). Several authors have advocated the advantages of diagrammatic representations with respect to sentential ones (see (Glasgow *et al.* 1995) for an account). However, because of the implicit, unalterable structure of the relations that they encode, analogical and diagrammatic representations tend to be less flexible and more domain specific than propositional ones. Following Barwise and Etchemendy, who believe that efficient reasoning is "inescapably heterogeneous" (Barwise & Etchemendy 1995, p.212), this paper has adopted a hybrid, model-based approach to knowledge representation, and presented (1) a new, expressive diagrammatic representation for planning based on setGraphs, structures that have the ability to implicitly capture the inherent spatial and topological constraints of a domain, and (2) a theoretical framework for hybrid planning based on (1), in which propositional and diagrammatic representations can be seamlessly integrated.

The work of Glasgow and Malton on model-based spatial reasoning (Glasgow & Malton 1994) contained many of the ideas adopted in the present framework, but it lacked a formalisation of the mapping between propositional and analogical structures. Glasgow and Malton proposed a knowledge representation for model-based 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. For example, the description – *The ball and the lamp are on the table; the lamp is to the right of the ball* – could be represented as the array
>
> | | |
> |---|---|
> | *ball* | *lamp* |
> | *table* | *table* |
>
> where the symbols *lamp*, *ball* and *table* are mapped to the corresponding entities in the world and the spatial relations of interest are *on-top-of* and *right-of*." (Glasgow & Malton 1994, p.7).

Glasgow and Malton provided a semantics for their representation by requiring that for a world to be represented by an array there must be a mapping between symbols in the array and entities in the world 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 $\mathcal{A}$ by the corresponding function ψ_i when $\psi_i(s_1,\ldots, s_n)$ returns '*true*' if and only if $(s_1,\ldots, s_n) \in r_i$ (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 ψ_i. The framework is further extended with the definition of possible, determinate and indeterminate worlds, and with a set of primitive *transformation* functions that can be used to manipulate arrays.

The implemented, Cartesian version of the setGraph model described earlier can be considered as a 'dynamic' version of the above representation. Glasgow and Malton's

model, however, lacked the formalisation of a paradigm for *describing* and automatically *reasoning about* the effects of an action on array models. By providing a syntax (namely, that of setGraphs) that allows the designer to specify inspection and transformation procedures declaratively, the present work generalises and extends that of Glasgow and Malton, allowing planning algorithms to be applied directly and effectively to spatial reasoning domains.

Myers and Konolige (Myers & Konolige 1995) described 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 consist 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 provided 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 roughly equivalent to the inspection procedures Ψ used in setGraphs. Reflection procedures, on the contrary, do not have a direct equivalent in setGraphs, and allow a diagram containing *structural uncertainty* to be transformed into a diagram that is (strictly) more determined by updating it with information obtained from the sentential deductive process. Unfortunately, Myers and Konolige's model did not include "retraction" procedures for *removing* existing analogical information from the diagram models. This appears to be crucial for allowing non-monotonic changes of a diagram, which are usually implied by the execution of an action. Hence, although Myers and Konolige's framework seems to be compatible with setGraphs (it should be possible to describe any setGraph as a set of analogical and label relations), it lacks a formal paradigm for *describing* and reasoning about the effects of *action* on diagram models. Further extensions to their model would be required before the results of their theoretical analysis can be used in a planning context. Similar considerations apply also to other works on heterogeneous representations, such as (Barwise & Etchemendy 1998) and (Swoboda & Allwein 2002).

The work of Forbus and colleagues on qualitative spatial reasoning (Forbus 1995) (Forbus, Nielsen, & Faltings 1987, 1991) is also relevant in this context. Forbus *et al.* propose 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. The advantage of setGraphs is that they allow a single, unified formalism of representation, in which qualitative and quantitative information are integrated and support both types of reasoning (without requiring the use of a complete, underlying metric diagram).

Within the area of planning research, the seminal work of Long and Fox (Long & Fox 2000) constitutes an interesting link with the present framework. Long and Fox have

developed domain analysis techniques that allow the automatic detection and exploitation of generic types of objects in a domain (such as 'mobiles' and 'portables') from its purely propositional description. Many domains are isomorphic to and can be treated as 'transportation' or 'construction' problems, even when this is not apparent from their description (see (Long & Fox 2000)). If the dynamics a domain can be recast in terms of movement or manipulation of (possibly abstract) objects (notice that any object *state-change* can be easily represented in terms of an 'abstract' movement (Garagnani 2003)), the constraints that restrict the movement of such objects can be made *implicit* in the domain description using analogical (setGraph) models, and significantly simplify and speed-up the planning process. 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. In addition, in those cases where a purely propositional analysis may not be able to reveal the equivalence of two structures (e.g., the "Monster Change" (Hayes & Simon 1977) and the Tower of Hanoi problems), an analogical encoding might constitute a more suitable representation for detecting the presence of such an isomorphism.

Although setGraphs can ultimately encode any PDDL2.1 domain (level 2), they can do so only if conditional effects, quantification and disjunctive preconditions are previously compiled away. In this sense, the expressiveness of the setGraph planning formalism presented is still limited. However, given the low-level mapping between a ground action schema and an instantiated setGraph action, it should be possible to extend the language to allow such features.

As mentioned earlier, 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. In addition, 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 normal setGraph places, while *transitions* (active nodes) can be represented as a specific type of node (let us call it '*Trans*'). Figure 10 shows a setGraph action schema encoding the movement of a *single* token in any Petri net (notice the addition of the symbol "∅" to the formalism, forcing a place that contains it to match only empty places). The simulation of the parallel movement of several tokens can be represented using similar action schemata.

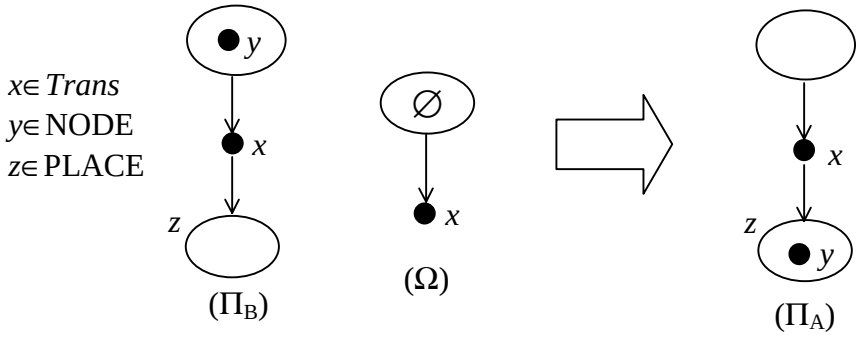

Figure 10. Petri net dynamics encoded by a setGraph action schema. Negative precondition (Ω) guarantees that all inputs to *x* contain at least 1 'token'.

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 present definition of setGraph. However, it should not be too difficult to extend setGraphs to allow this feature, and obtain even more expressive planning domain-description formalisms.

In addition to avoid the problem of ramification and improve performance, the adoption of analogical representations should also allow more effective common-sense reasoning, supporting more efficient learning, heuristic extraction and abstraction techniques through new reasoning modes that are not possible (or not practical) in purely propositional languages. For example, consider the task of determining the existence of identical stacks of blocks in two different BW states: two setGraph models can be efficiently checked and 'matched' against each other in search of recurring patterns of places and nodes. Resource production and consumption can also be easily represented using setGraphs. Furthermore, setGraphs allow more intuitive and direct description languages, leading to less error-prone domain encodings and better modelling.

Nevertheless, the formalism proposed is by no means claimed to be a *universal* language for knowledge representation or reasoning about action. The main aim of this document was to lay the foundations for enabling the development of more efficient, sound, hybrid (or analogical) domain-description languages and planners, based on setGraphs or on more expressive, non-sentential structures that extend them. In addition to having achieved the original goal set, the author also hopes that this work will stimulate co-operation and exchange between the planning, knowledge representation and common-sense reasoning communities, encouraging researchers to work across the boundaries of these closely related areas and leading to the growth of new branches of research.

References

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

Barr, A. and Feigenbaum, E. A. eds. (1981) *The Handbook of Artificial Intelligence. Vol. 1*, HeurisTek Press, Stanford, CA and William Kaufmann, Los Altos, CA.

Barwise, J. and Etchemendy, J. (1995) Heterogeneous Logic. In (Glasgow *et al.* 95), chapter 7, pp. 211–234.

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

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

Davidson, M. and Garagnani, M. (2002) Pre-processing planning domains containing Language Axioms. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling SIG (PlanSIG 2002*), pp. 23–34. Delft (NL).

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

Fikes, R.E. and 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., and Faltings, B. (1987) Qualitative Kinematics: A Framework. In *Proceedings of the International Joint Conference on Artificial Intelligence*, San Francisco, CA. Morgan Kaufmann.

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

Fox, M. and Long, D. (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research - Special issue on the 3rd International Planning Competition* (forthcoming).

Garagnani, M. (2000) A correct algorithm for efficient planning with preprocessed domain axioms. In Bramer, M., Preece, A. and Coenen, F., eds. *Research and Development in Intelligent Systems* XVII, pp. 363–374. Springer-Verlag.

Garagnani, M. (2003) Model-based Planning in Physical domains using SetGraphs. In *Proceedings of the 23rd International Conference on Innovative Techniques and Applications of AI (AI-2003)*, Cambridge (UK), Dec.'03 (forthcoming).

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

Gazen, B.C. and Knoblock, C.A. (1997) Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In *Proceedings of the 4th European Conference on Planning* (*ECP-97)*, pp. 221–233. Toulouse, France.

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

Giunchiglia, F. and Traverso, P. (1999) Planning as model checking. In Biundo, S. and Fox, M., eds.: *Proceedings of the 5th European Conference on Planning (ECP-99)*, pp. 1–20.

Glasgow, J., and 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 (1994).

Glasgow, J., Narayanan, N.H., and Chandrasekaran, B., eds. (1995) *Diagrammatic Reasoning*. AAAI Press / The MIT Press, Cambridge MA.

Halpern, J.Y. and Vardi, M.Y. (1991) Model Checking vs. Theorem Proving: A Manifesto. In Allen, J. A., Fikes, R. and Sandewall, E. (eds.), *Principles of Knowledge representation and Reasoning: Proceedings of the 2nd International Conference (KR91)*, pp. 325–332.

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

Hayes, P.J. (1974) Some problems and non-problems in representation theory. In *Proceedings of the AISB Summer Conference*, pp. 63–79. University of Sussex, Brighton, England.

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

Khardon, R. and 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*. Stanford, CA: AAAI Press.

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

Larkin, J.H. and 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, pp. 69–109.

Levesque, H. (1992) Is reasoning too hard? In *Proceedings of the 3rd NEC Research Symposium*.

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

Lifschitz, V. (1986) On the semantics of STRIPS. In Georgeff, M.P. and Lansky, eds., *Proceedings of 1986 Workshop: Reasoning about Actions and Plans*.

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

Long, D. and Fox, M. (2000) Automatic synthesis and use of generic types in planning. In Chien, S., Kambhampati, S., Knoblock, C., eds.: *Proceedings of the 5th International Conference on AI Planning and Scheduling Systems (AIPS'00)*, pp.196–205. Breckenridge, CO: 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 Minsky's (ed.) *Semantic Information Processing*, pp. 403–409. MIT Press (1968). Also in Brachman, R. and Levesque, H. (eds.), Readings in Knowledge representation (1985).

McCarthy, J. and Hayes, P.J. (1969) Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B. and Michie, D. (eds.) *Machine Intelligence* ***4***. Edinburgh: Edinburgh University Press.

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

Myers, K. and 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. and Lloyd, B.B. (eds.), *Computing and Categorization*, pp. 259–303. Hillsdale, NJ:Earlbaum.

Pednault, E. (1988) Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* ***4***:356–372.

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

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

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

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)*, pp. 164–171. Cambridge, MA.

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

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

Thiébeaux, S., Hoffmann, J. and Nebel, B. (2003) In Defense of PDDL Axioms. In *Proceedings of ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*. Trento (Italy).

Winslett, M. (1988) Reasoning about action using a possible models approach. In *Proceedings of the 7th National Conference on AI (AAAI-88)*, pp. 89–93.