A Diagrammatic Meta-Language for Planning Domain Descriptions

Max GARAGNANI*
Department of Computing, The Open University
Walton Hall, Milton Keynes MK7 6AA – U.K.
Tel. +44 (0)1908 654812 Fax. +44 (0)1908 652140
M.Garagnani@open.ac.uk

Abstract. Sentential and diagrammatic representations are two different formalisms for describing domains and problems. Sentential descriptions are usually more expressive than diagrammatic ones, but tend to present a more complex and less intuitive notation. All modern planning domain description languages are sentential. The complexity of sentential formalisms has been of hindrance to the wider dissemination and take up of planning technology beyond the planning research community. This paper proposes a diagrammatic "meta-language" for planning domain descriptions based on setGraphs as an alternative to sentential languages. SetGraphs represent actions, states and goals in terms of set- and graph-theoretic constructs. Through various practical examples, setGraphs are shown to yield simpler and more intuitive domain encodings, and to offer a high degree of elaboration tolerance. A theoretical analysis shows how the representation can be easily encoded using formal languages, and demonstrates that setGraphs are at least as expressive as a standard modern propositional planning domain description language. The model proposed constitutes a "core" representation that can be adopted as a basis for developing different planning domain description languages; it is suitable to be used across different *levels of abstraction* during the processes of language development and domain knowledge engineering, and it facilitates the elicitation, maintenance and re-use of planning domain descriptions.

1 Introduction

During the past few years, research in AI planning has reached a stage of maturity that should have resulted in wide industrial and commercial take up. However, this has not generally happened. One of the causes of the gap between theory and applications lies in the fact that the remarkable capabilities of modern planning technology (and, perhaps, even the *existence* of the technology itself) are still largely ignored outside the planning community. This is not hard to believe, given the relatively young age of the field and the speed at which it has been developing. Indeed, many AI researchers who are not directly involved in planning research still consider this area as mostly concerned with STRIPS-like systems and problems.

In order for planning technology to find wider application, more awareness and a clearer understanding of what planning currently is and what it has to offer are required. The rather specialist, logic-based notation that all modern domain description languages adopt appears

*The author wishes to thank the UK EPSRC for the support given to this research (grant ref. GR/R53432/01).

to be one of the major obstacles to the technology dissemination and take up, particularly in non-academic environments. In fact, although AI practitioners are (usually) familiar with the concepts of states and goals and are well versed in predicate logic, potential users, knowledge engineers and domain experts tend to be discouraged when presented with the latest BNF formalisation of PDDL [12], and when required to express their problem in such terms. Paradoxically, in several cases PDDL still fails to offer the flexibility and expressiveness that is needed to accurately and completely describe the real problem at hand anyway.

In order to allow planning and non-planning experts to more easily represent, communicate and modify planning domain descriptions, we propose a *diagrammatic* (or analogical [13]) formalism based on the model of *setGraph* [4, 6, 5]. SetGraphs represent states, goals and actions in terms of directed graphs in which the vertices are sets of symbols. SetGraphs were introduced in [4] as a simpler and more efficient alternative to traditional, pervasive, *sentential* domain description languages. SetGraphs representations can be easily translated into more formal domain description languages. Furthermore, preliminary experimental evidence [6] suggests that planning problems represented in diagrammatic terms can be solved much faster (up to two orders of magnitude) than when described using a purely sentential language. This paper argues that setGraphs also yield simpler and more intuitive (yet flexible) domain descriptions, which can be more easily and quickly understood by non-planning experts and facilitate the process of domain knowledge engineering.

The paper is organised as follows: Section 2 introduces the main ideas behind setGraphs through simple examples; Section 3 presents setGraphs in more precise and formal terms, introducing numeric values in the model; in Section 4, setGraphs are proven to be expressively equivalent to a simplified version of the standard planning domain description language PDDL. Related work, limitations and future directions are discussed in the last section.

Before proceeding any further, it should be clarified that setGraphs are not claimed here to be “the” universal language which is fit for *all* application domains. Adopting the same view of Frank, Golden and Jonsson [3], we propose setGraphs as a common “core” representation for use in many planning domain description languages, sufficiently expressive for most simple, well-defined, problems, and easy to be extended to meet the different requirements of the various real-world situations. In addition, it should be pointed out that this paper is not concerned with the definition of a specific language for the computational encoding of the proposed diagrammatic representation. The representation described here constitutes a “meta-language”, aimed at facilitating the processes of domain description language development and knowledge engineering, including domain knowledge elicitation, representation, modification and re-use. The task of specifying the actual syntax of a domain description language based on setGraphs, although relatively straighforward (setGraphs are built using basic concepts of set and graph theory), does not fall within the scope of this work[1].

2 A gentle introduction to SetGraphs

We start with a very simple representation, which is then progressively augmented with additional features. The formal description of the formalism is given in the next section.

A setGraph is essentially a directed graph in which the vertices are sets of symbols. For example, Figure 1.(a) shows a setGraph description of a Blocks World (BW) state with three

[1]However, see [6] for an example of BNF formalisation of a simple language based on a restricted version of the setGraph model.

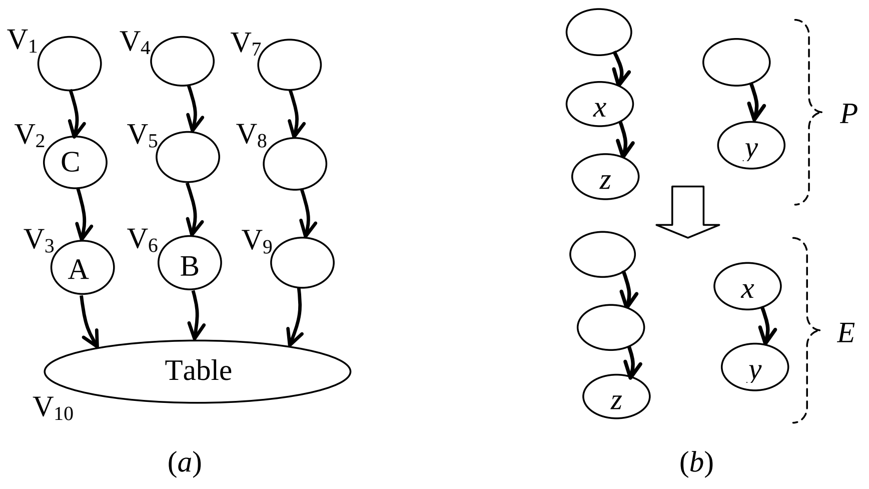

Figure 1: A SetGraph encoding of the Blocks World domain: (a) state representation; (b) *Move*(x, y, z) operator (where $x \in \{$A, B, C$\}$ and $y, z \in \{$A, B, C, Table$\}$).

blocks and a table, represented by symbols ‘A’, ‘B’,‘C’, ‘Table’. The vertices of the graph are depicted as ovals, labelled V$_1, \ldots$,V$_{10}$. In this example, the edges of the graph (arcs) represent ‘on’ relations between spatial locations: if a vertex containing symbol x is linked to a vertex containing symbol y, then `On` (x, y) holds in the current state.

The symbols of a setGraph can be moved from one vertex (set) to any other through the application of diagrammatic operators, which specify the set of legal transformations of a state (setGraph). Figure 1.(b) depicts the *Move* operator for the BW domain. The operator preconditions P describe a specific arrangement of symbols in a part (sub-graph) of the current state; the effects E describe the arrangement of these symbols in the same sub-graph after the application of the operator. Intuitively, an operator $P \Rightarrow E$ is applicable in a state s iff each of the graphs contained in P can “overlap” with (be mapped to) a sub-graph of s having the same “structure”, so that each variable corresponds to a distinct symbol, each vertex to a vertex, each edge to an edge, and: (1) if a variable is contained in a set (vertex), the corresponding symbol is contained in the corresponding set; (2) if an edge links two vertices, the corresponding edge links the corresponding sets; and (3) if a set is empty, the corresponding image is empty. When all of these conditions hold, we will say that the precondition setGraphs are *satisfied* in s. Notice that the variables can be of specific *types*, subsets of the universe of symbols. For example, variable x of the *Move*(x, y, z) operator has type *Block*=$\{$A, B, C$\}$, while $y, z \in$ *Object*=$\{$A, B, C, Table$\}$. Thus, this operator encodes the movement of a block x from its current location to a new one, originally empty, situated “on top” of a set containing another block (or the table) y. Notice that the operator is applicable only if block x has an empty set on top of it (i.e., if x is clear).

The application of an operator to a state s causes the corresponding symbols in s to be rearranged according to the situation described in the effects E. For example, the *Move*(x, y, z) operator can be applied to the state of Figure 1.(a) in several ways. One possible binding is x/C, y/Table, z/A; the application of *Move*(C,Table,A) would unstack block C from A and put it on the table (i.e., in set V_9 of Figure 1).

This simple representation can be easily extended with additional features. The following example illustrates the use of typed and symbol-to-symbol edges in setGraphs; in the next

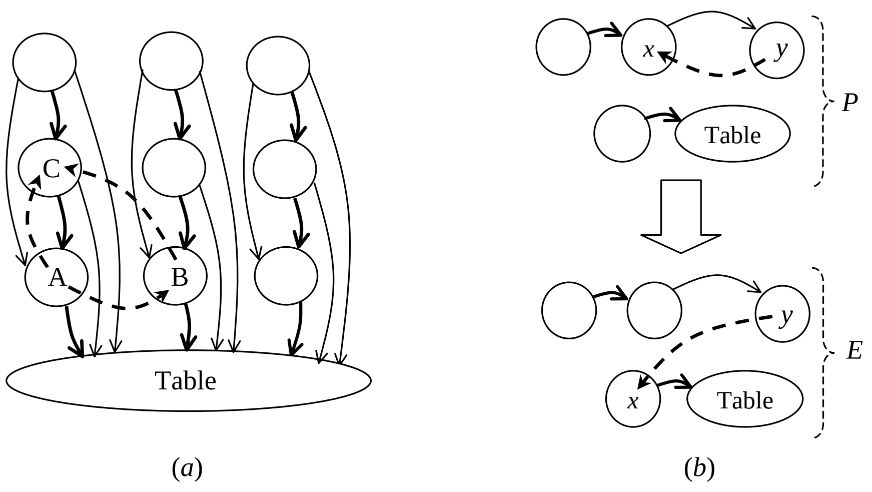

Figure 2: A richer diagrammatic model of BW: (a) current state; (b) *Unstack*(x, y) operator. Bold, thin and dashed arcs indicate, respectively, 'on', 'above' and 'heavier' relations (see Example 1 for details).

section, symbol-vertex edges and numeric quantities will be introduced. As discussed later, the addition of these features makes the formalism significantly more expressive.

Example 1 - Consider a variation of BW in which blocks have a specific weight, and the stacks are such that a block can be removed from the top only if the stack contains at least another block which is *heavier* than the one being removed. In order to encode this domain, two additional relations between objects (namely, the relation 'heavier' and 'above' between blocks) have to be represented. One possible way to obtain this is to introduce typed edges. Figure 2.(a) depicts different types of edges using different *styles* of arcs (although *labelled* arcs could also have been used). The state representation contains two new types of edges, identified by thin and dashed arcs. A thin arc linking a vertex containing symbol x to one containing symbol y denotes that block x is the n-th block above y (with $n > 1$); a dashed arrow linking symbol x to symbol y indicates that block x is heavier than block y. Figure 2.(b) depicts the diagrammatic version of the *Unstack*(x, y) operator (the *Stack*(x, y) operator is identical to the *Move*(x, y, z) operator of Figure 1.(b) with z=Table). The preconditions P require the existence of a block y such that both `Heavier`(y, x) and `Above`(x, y, n) hold. The effects E encode the new position of block x, now located on the table. We assume that when a symbol is moved, all arcs connected to it move with it.

Notice that the number of edges needed in a setGraph to represent relations between objects (like 'above' or 'heavier') grows only *polynomially* in the number of objects of the domain. Another interesting property of diagrammatic representations is that they allow the spatial relations existing between the "mobile" objects of a domain to be represented as *static* (or invariant) elements of the description, whenever the set of possible positions in which such objects can be (relatively to each other) is finite and predetermined. For example, it is easy to see that all the edges of the setGraph of Figure 2.(a) (including all those representing spatial relations between blocks) are static and remain unchanged throughout any plan execution, as they are not affected by any of the actions. Interestingly, this encoding could have been also "emulated" using a propositional language description. For example, if the vertices of the setGraph of Figure 2.(a) were considered as entities of the domain, then the BW state could be described also as follows:

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

Predicate `In`(x,y) indicates that symbol x is inside vertex y; `Empty`(x) indicates that vertex x contains no symbols; `On`(x,y), `Above`(x,y) and `Heavier`(x,y) denote the corresponding edges between vertices and symbols. Observe that in the state representation I, only the instances of the predicates `In`(x,y) and `Empty`(x) are subject to change; the rest of the literals are static. Notice that, although obtaining this propositional encoding from Figure 2.(a) appears to be a straightforward task, no planner would have been able to *automatically* generate state I from the original, propositional description of the BW domain.

Whilst this graph-based notation is capable to encode most of the "classical" benchmark domains, a more expressive formalism is required for most real-world domains. The next section contains a more precise and formal definition of setGraphs, in which the representation is further extended with numeric quantities and actions involving not only the movement (or removal) of symbols but also of vertices. As pointed out earlier, however, the main aim of this paper is not to build a domain description language suitable for encoding all possible real domains, but rather to introduce a common core, simple and intuitive diagrammatic formalism, easy to understand and use, which can be easily extended with additional features according to the different requirements of the situation. In view of this, the description of the model has been kept intentionally abstract and stripped of any low-level syntactical detail, so as to avoid committing the representation to any particular language or paradigm.

3 Formalising setGraphs

In order to formally define a setGraph, let us introduce the *collection* construct. A collection is a data structure identical to a list, except that the order of the elements is unimportant. Equivalently, a collection can be seen as a set in which multiple occurrences of the same element are permitted. Notice that the multiple instances of an element should be thought of as distinct elements of the structure. For example, C=$\{1,1,0,0,0\}$ denotes a collection of integers containing two occurrences of the number 1 and three occurrences of the number 0. Since the order is unimportant, any permutation of the elements of C is equivalent to the same collection. Hence, C=$\{1,0,1,0,0\}$=$\{1,0,0,1,0\}$=$\{1,0,0,0,1\}$=$\{0,1,0,1,0\}$=.... The empty collection is denoted as $\{\ \}$. We will write "$x \in$ C" and say that x is *contained* in C to indicate that element x appears (occurs) at least once in collection C.

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

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

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

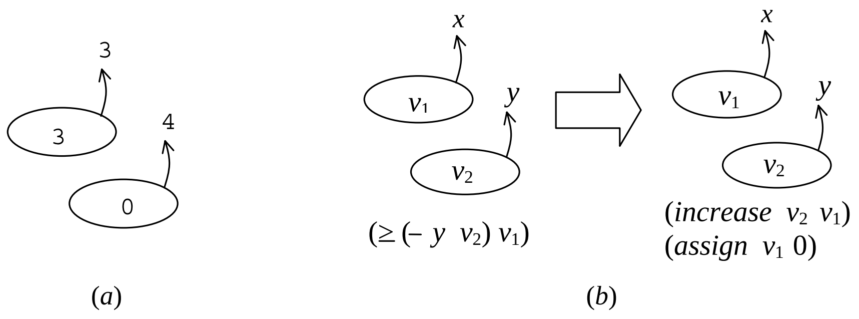

Figure 3: A diagrammatic model of the Water Jug domain: (a) initial state; (b) *Pour* operator.

A node is a symbol (string) of the language. A place is a "container" for both nodes and places. Nodes and places are nodeSets. In short, nodeSets are data structures consisting of multi-nested sets of symbols (strings) with multiply occurring elements and no limit on the level of nesting. Places can be labelled (possibly with identical labels). Given a nodeSet N, $\wp(N)$ is defined as the collection of all the nodeSets occurring in N (including N itself). For example, consider a language W={A,B,C}. Let N_1 be the nodeSet {A, {B}, {{C}}}. Then, $\wp(N_1)$= {A, B, C, P$_0$, P$_1$, P$_2$, P$_3$}, where P$_0 = N_1$, P$_1 = \{B\}$, P$_2 = \{\{C\}\}$ and P$_3 = \{C\}$.

Definition 2 (setgraph). *A* setGraph *is a pair $\langle N, E\rangle$, where N is a nodeSet and $E = \{E_1, \dots, E_k\}$ is a finite set of binary relations on $\wp(N)$ (i.e.,$E_i \subseteq \wp(N) \times \wp(N), i \in \{1, ...k\}$).*

If E contains only one relation E', we shall simply write $\langle N, E'\rangle$ instead of $\langle N, \{E'\}\rangle$. For example, let N_1 be the nodeSet N_1={A,{B},{{C}}}. The pair $\alpha = \langle N_1, E_1\rangle$ is a setGraph, where E_1={(C, B), ({B}, {{C}}), ({A,{B},{{C}}}, A)}. The instances of the binary relation E_1, pairs of elements of $\wp(N_1)$, are the edges of the setGraph. Notice that if N_1 is specified using the notation N_1=P$_0${A, P$_1${B}, P$_2${P$_3${C}}} in which all places are labelled, then E_1 can be specified simply as {(C, B), (P$_1$, P$_2$), (P$_0$, A)} (refer to the previous example).

A complete treatment of the setGraph representation lies outside the scope of this paper, and can be found in [5]. The above description, however, illustrates how setGraph models can be easily formalised using set- and graph-theoretic constructs.

3.1 Actions with numeric and non-conservative effects

The simple planning operators introduced in Section 2 were limited to the movement of nodes from one place to another. Let us now augment the representation so as to allow the description of actions containing numeric preconditions and effects, and involving the movement, addition to, or removal from the setGraph of any element (nodeSet or edge).

Numbers are represented in setGraphs as *numeric* nodes. A numeric node is a string (symbol) of W of form $n.m$ or n (possibly preceded by $+$ or $-$), where n, m denote sequences of digits. Operators preconditions and effects will consist of two separate parts, diagrammatic and numerical. The diagrammatic preconditions and effects are lists of parameterised setGraphs. The numerical preconditions consist of a set of comparisons $(<, >, =, \leq, \geq, \neq)$ between pairs of numerical expressions. Numeric expressions are built from the values of numeric nodes using arithmetic operators. Numeric effects can make use of a set of assignment operations that update the values of a numeric node; these include direct assignment and relative assignments (such as increase and decrease – see Figure 3 and example below).

```
(define (domain jug-pouring)
   (:requirements :typing :fluents)
   (:types jug)
   (:functions
        (amount ?j - jug)
        (capacity ?j - jug))

   (:action pour
       :parameters (?jug1 ?jug2 - jug)
       :precondition (>= (- (capacity ?jug2) (amount ?jug2))
       (amount ?jug1))
       :effect (and (assign (amount ?jug1) 0)
                    (increase (amount ?jug2) (amount ?jug1)))
   )
)
```
Figure 4: PDDL2.1 encoding of the Water Jug problem (*from* [2]).

The possible setGraph transformations considered here are: (i) addition or removal of a node, place or edge, (ii) movement of a nodeSet, and (iii) update of a numeric node. The movement and removal of elements in a setGraph is based on the following general rules: (1) if a node is moved (removed), all edges linked to it move (are removed) with it; (2) if a place is moved (removed), all the elements contained in it and all edges linked to it move (are removed) with it. Any element not moved, removed or updated is left unaltered.[2]

Example 2 - In the well-known Water-Jug domain, the action *Pour* allows the water in one jug to be emptied into a second jug, provided that the capacity of the second jug is sufficiently large to hold the total volume of water currently present in the two jugs. One possible way to describe this domain using setGraphs consists of considering each jug as a place, *associated to* (i.e., linked by an edge to) a numeric node. The value of the node denotes that jug's capacity. Each jug will also contain a numeric node, representing the current volume of water in it. Figure 3.(a) encodes a possible initial state for this problem, in which two jugs of capacity 3 and 4 contain, respectively, 3 and 0 units of water. Figure 3.(b) encodes the *Pour* operator. This action has no effect on the arrangement of the nodes and places in the setGraph, as it only updates the numeric values of nodes v_1 and v_2. Figure 4 contains the sentential description of the Water-Jug domain, expressed in PDDL2.1 style [2]. When compared to sentential descriptions, diagrammatic representations appear to be simpler and more natural, less error-prone and easier to modify and re-use. As demonstrated in the next section, setGraph representations are also at least as expressive as modern propositional planning domain description languages.

4 Expressiveness of setGraphs

The expressiveness of setGraphs is assessed by comparing the representation against a propositional formalism that allows the use of numeric quantities and functor symbols. In particular,

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

we adopt the same semantic model adopted in PDDL2.1 [2]. In short, the PDDL2.1 semantics is based on the idea of a state being composed of two separate parts, a *logical* state and a *numeric* state. While a logical state L consists of a set (conjunction) of ground atomic formulæ (the truth of an atom p depending on whether $p \in L$), the numeric state consists of a (finite) vector R of values in $\Re_{\perp} = \Re \cup \{\perp\}$, representing the complete set of *primitive numeric expressions* (PNEs) of the problem (values associated with tuples of domain objects by domain functions – see [2] for a more detailed explanation). An operator specifies a transformation of a state-pair $s = (L, R)$ into a new state-pair $s' = (L', R')$. In this analysis we consider *ground* (i.e., instantiated) operators of form $(P \Rightarrow E)$, in which the preconditions P contain a list of ground literals and a set of comparisons between pairs of numeric expressions (containing PNEs and numbers), while the effects E consist of list of ground literals and a set of numeric update operations (analogous to those adopted in setGraphs). Notice that any PDDL2.1 ("level 2", i.e., non-durative) action schema can be compiled into an *equivalent* set of (ground) operators of the above form [2].

The two main results concerning the expressiveness of setGraphs are presented below:

Theorem 1 (Expressiveness). *Every propositional* (PDDL2.1) *state* $s = (L, R)$ *can be represented as a setGraph.*

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

Corollary *Every ground propositional (*PDDL2.1*, level 2) operator can be transformed into an equivalent setGraph operator.*

Proof sketch Given the encoding used in the proof of the above theorem, every sentential operator can be transformed into an equivalent setGraph operator as follows: each addition (removal) of an atom $p(x_1, \ldots, x_n)$ to (from) the logical state L corresponds to the addition (removal) of the corresponding node "p" and associated edges to (from) place *Pred*. Similarly, each update of a PNE $f(x_1, \ldots, x_m)$ in R is encoded through the update of the numeric node w at the end of the chain $(f, x_1), (x_1, x_2), \ldots, (x_m, w)$. *Q.E.D.*

Notice that the transformation of sentential descriptions into setGraph models is *polynomial* (while the size of the result is *linear*) in the size of the original encoding[3].

[3]The measure of this size includes the size of R and the arity of the PNEs and predicates of the language.

5 Related work and Discussion

The aim of this paper was to propose a diagrammatic meta-language for planning domain descriptions, which would facilitate the process of language development and domain knowledge engineering. The examples illustrated that setGraphs produce simpler and more intuitive domain descriptions, easier to understand for non-planning experts. This should allow researchers and practitioners outside the planning community to better understand modern planning technology, facilitating its dissemination and take-up, and hence reducing the existing gap between theory and real-world applications. At the same time, the setGraph formalism described was proven to be at least as expressive as a propositional language that allows the use of numeric quantities and functor symbols, and to be easily modelled using more formal, set- and graph-theoretic constructs. These characteristics make the model suitable to be used across different levels of abstraction (e.g., as a tool for knowledge elicitation and exchange between planning experts and domain experts, or, alternatively, as a basis for domain description language development). In particular, setGraphs can be used as a common core representation – in other words, as an underlying "platform" – for the development of different domain description languages having different syntax and/or additional components (see the requirements for a core language proposed by Frank *et al.* in [3]).

In terms of flexibility and capability to undergo extensions, setGraphs display a high degree of *elaboration tolerance*. This was illustrated, for example, when a new, revised version of the BW domain was introduced (Figure 2): the setGraph representation used to encode the new domain completely subsumed and re-used the encoding adopted for the original version (Figure 1). In other words, the additional requirements (relationships 'heavier' and 'above' between blocks) simply led to the old representation being augmented with new elements.

The use of models that reflect the spatial and topological aspects of the real domain suggests that this type of representation should also better support common-sense reasoning, abstraction, heuristic extraction and learning techniques. In fact, observe, for example, that the ability to automatically learn control knowledge through repeated problem solving in the same domain depends heavily on the system being capable to recognise common "patterns" and structures in different plan solutions. Consider the complexity of identifying such patterns in sequences of propositional states (e.g., think of determining the existence of two identical stacks of blocks in different BW states): sets of places and edges can be much more efficiently compared and "overlapped" in search of the occurrence of the same pattern.

SetGraphs borrow ideas from various related works in the area of analogical representations [7, 9], qualitative reasoning [1], set and graph theory [8] and semantic networks (in particular, conceptual graphs [14]). However, it appears that the use of diagrammatic representations in *planning* had never been really investigated before. The work of Fox and Long on generic types [11], however, fits particularly well with the present approach. Preliminary experimental evidence (reported in [6]) indicates that problems involving the movement of objects subject to constraints can be solved significantly faster (up to *two* orders of magnitude) when recast in diagrammatic terms. The work of Fox and Long suggests that many domains are actually *isomorphic* to and can be treated as "transportation" or "construction" problems [11]. If the dynamics a domain can be recast in terms of movement or manipulation of objects, setGraphs and diagrammatic representations may be adopted to encode and solve problems in this domain more efficiently.

Two important aspects of action modelling that have not been dealt with here concern the specification of *concurrent* and *durative* (analogical) actions. A possible approach to identifying non-interference conditions for the concurrent execution of setGraph operators may be to require that the elements of the setGraph model upon which the operators act be disjoint. However, the introduction of time and durative actions, possibly in presence of other features – such as conditional and continuous effects – makes this a rather complex issue, requiring further investigation.

Finally, it should be mentioned that a general theoretical framework that identifies the conditions for the *soundness* of setGraph, diagrammatic and *hybrid* representations has been described in [5]. These conditions extend those identified by Lifschitz in [10] for sound action, originally limited to sentential models and still at the basis of all current sentential planning description languages [2].

References

[1] K. Forbus. Qualitative Spatial Reasoning: Framework and Frontiers. In *[7]*, chapter 6, pages 183–202. 1995.

[2] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research,* 20:61–124, 2003.

[3] J. Frank, K. Golden, and A. Jonsson. The loyal opposition comments on plan domain description languages. In *Proceedings of 13th International Conference on Automated Planning and Scheduling (ICAPS'03) - Workshop on PDDL*, pages 39–48, Trento, Italy, June 2003.

[4] M. Garagnani. Model-Based Planning in Physical domains using SetGraphs. In M. Bramer, A. Preece, and F. Coenen, editors, *Research and Development in Intelligent Systems XX (Proc. of AI-2003),* pages 295–308. Springer-Verlag, December 2003.

[5] M. Garagnani. A Framework for Hybrid and Analogical Planning. In I. Vlahavas and D. Vrakas, editors, *Intelligent Techniques for Planning*. Idea Group, Inc., 2004. (to appear).

[6] M. Garagnani and Y. Ding. Model-based planning for object-rearrangement problems. In *Proceedings of 13th International Conference on Automated Planning and Scheduling (ICAPS'03) - Workshop on PDDL*, pages 49–58, Trento, Italy, June 2003.

[7] J. Glasgow, N.H. Narayanan, and B. Chandrasekaran, editors. *Diagrammatic Reasoning*. AAAI Press/The MIT Press, Cambridge, MA, 1995.

[8] D. Harel. On Visual Formalisms. *Communication of the ACM*, 13:514–530, May 1988.

[9] Z. Kulpa. Diagrammatic representation and reasoning. *Machine GRAPHICS & VISION*, 3(1/2):77–103, 1994.

[10] V. Lifschitz. On the semantics of STRIPS. In M.P. Georgeff and Lansky, editors, *Proceedigns of 1986 Workshop: Reasoning about Actions and Plans*, 1986.

[11] D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In S. Chien, S. Kambhampati, and C.A. Knoblock, editors, *Proceedings of the 5th International Conference on AI Planning and Scheduling (AIPS'00),* pages 196–205, Breckenridge, CO, April 2000. AAAI Press.

[12] D. McDermott, C. Knoblock, M. Veloso, D. Weld, and D. Wilkins. PDDL – the planning domain definition language. Version 1.7. Technical report, Department of Computer Science, Yale University (CT), 1998. (Available at www.cs.yale.edu/homes/dvm/).

[13] K. Myers and K. Konolige. Reasoning with analogical representations. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92),* pages 189–200. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[14] J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, MA, 1984.