Technical Report No: 2004/04

A Framework for Hybrid Planning

Max Garagnani

19th February 2004

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

http://computing.open.ac.uk

A Framework for Hybrid Planning

Max Garagnani
Department of Computing, The Open University - MK7 6AA (UK)
M.Garagnani@open.ac.uk

Abstract. Hybrid models are characterised by the integration of two (or more) different paradigms of representation within the same system. Most current planning problem description languages are purely *sentential*, i.e., based on predicate logic formalisms [3, 14]. The *ramification* of the effects of action [16] makes the sentential description of planning problems that involve the spatial movement of physical entities over-complex and inefficient. Evidence from research in knowledge representation and planning [6, 9, 8] indicates that these problems are more effectively modelled using *homomorphic* [1] (or analogical) representations. As for sentential descriptions, however, the complexity of the real world prevents purely homomorphic formalisms from being *always* the most effective choice. This paper proposes a model-based theoretical framework for planning with hybrid representations, in which equivalent sentential and analogical models can be simultaneously and interchangeably used. The analogical model which is developed extends a recently proposed representation for purely homomorphic planning [5]. The sentential model of action adopted is based on the current standard planning domain description language PDDL2.1 [3]. The result is a powerful, heterogeneous planning representation that overcomes the limitations and offers the complementary strenghts of the two formalisms on which it relies.

1 INTRODUCTION

Evidence from research in knowledge representation and reasoning (see [9, 8] for useful accounts) indicates that many problems become easier to solve if described using *homomorphic* [1] (or analogical, diagrammatic [17, 10]) representations. Recent experimental evidence in planning [6, 5] demonstrates that problems that require planning the spatial movement and manipulation of many (abstract or physical) entities are solved significantly faster (up to *two* orders of magnitude) if recast in analogical terms. Although more efficient, however, homomorphic representations are often criticised for their limited expressiveness, which restricts their use to only specific domains of application and prevents them from offering the universality of propositional languages.

The aim of this work is to develop hybrid (or *heterogenerous* [1, 18]) planning representations, able to merge sentential and analogical models into a single formalism that combines the strengths and overcomes the weaknesses of the two paradigms. In this paper, we (i) review the '*setGraph*' model described in [5, 6] (to date, the only existing proposal of homomorphic planning representation) and extend it into a more expressive representation (Section 2); (ii) briefly describe the *sentential* model chosen, based on the planning domain description language PDDL2.1 [3] and expressively *equivalent* to the analogical model (Section 3); (iii) describe a simple model of hybrid planning (Section 4) which allows the two above representations to be integrated; and (iv) present a general theory that guarantees the soundness of the approach (Section 5). In particular, the 'Soundness Theorem', presented at the end of Section 5, extends to analogical and hybrid representations the theory of *sound* action description (originally given in [11]), until now limited to sentential models. The final section discusses related work, limitations and future directions.

2 THE ANALOGICAL MODEL: SETGRAPHS

This section extends and recasts in more formal and rigorous terms the setGraph model proposed in [5]. The original model is extended to allow (1) *numeric values* (hence, attributes with infinite domains), and (2) actions involving *non-conservative changes* (addition and removal of elements to and from a state) and *numeric updates*.

2.1 Extending SetGraphs with Numeric Values

In essence, a SetGraph is a collection of *nodeSets*, defined as follows:

Definition 1 (NodeSet) *A* nodeSet *is either:*

- *a* node*, i.e., the empty-set element '$\emptyset$', or*
- *a* place *i.e., a finite set of nodeSets.*

Thus, nodeSets are 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. Notice the difference between a node (a constantly empty set '$\emptyset$'), and an empty place '$\{\ \}$' (a nodeSet which just happens to be empty). Given a nodeSet N, $\wp(N)$ represents the set of all the nodeSets contained in N (including N itself).

Definition 2 (SetGraph) *A* setGraph *is a pair $\langle N, E\rangle$, where N is a nodeSet and E=$\{E_1, ..., E_k\}$ is a finite set of binary relations ('edges') on $\wp(N)$.*

Originally, nodeSets could only be associated to labels. The following definition extends the formalism to allow also the use of numbers.

Let $\Re_\perp = \Re \cup \{\perp\}$, where $\Re$ is the set of real numbers, and $\perp$ is the 'undefined' value. An element of $\Re_\perp$ will be called an 'R-value'.

Definition 3 (Ground nodeSet) *A* ground nodeSet *is a nodeSet in which every place is associated to a label (string of characters) and every node is associated either to an R-value or to a label.*

Nodes, places and edges of a setGraph are grouped in different types (or sorts), specified using three hierarchies of labels having, respectively, "NODE", "PLACE" and "EDGE" as roots. Every type 't' represents the finite set of *instances* (leaves) of the subtree that has t as root. Different types may have different properties. The properties of a type are inherited by all of its subtypes and instances. In the following example, the PLACE hierarchy is used to constrain the type of nodeSets that select places will be allowed to contain.

Example 1 - Consider a simple 'Briefcase' scenario, consisting of two connected locations (home and office), one briefcase, and three objects (called 'A', 'C' and 'D'). The briefcase can be moved freely from one location to the other. The objects can only be put inside and taken out of the briefcase at one of the two locations. The state {`(at Brief Home) (at A Home) (at C Home) (in D Brief)`} can be encoded as a setGraph $\alpha = \langle P_1, R_1 \rangle$, where:

P_1 = { `Home{A,C,Brief{D}}`, `Off{ }`, 1 }
R_1 = { (`Home`,`Off`), (`Off`,`Home`), (`Brief`,1) }

The syntax "*name*$\{x, y, ..., z\}$" denotes a place with label '*name*' containing nodeSets $x, y, ..., z$; all nodes are simply represented by their associated label or R-value. Two (unlabelled) edges in R_1 are used to represent the bidirectional connection between the two locations. The node with value '1' represents the number of objects currently contained in the briefcase; this attribute is encoded through an edge associating place '`Brief`' to a (numeric) node.

Figure 1.(a) shows a graphical representation of setGraph α. All and only the nodeSets that are contained in a place are depicted within the perimeter of the corresponding oval. Figure 1.(b) contains the NODE and PLACE sort hierarchies (the EDGE hierarchy, not shown, has no other node than the root). Notice that nodes and places can be considered as subtypes of 'nodeSet'. The PLACE hierarchy restricts any instance of a '`Mobile`' place (in this case, '`Brief`') to contain only '`Portable`' nodes (by default, a place would be allowed to contain any PLACE ∪ NODE instances).

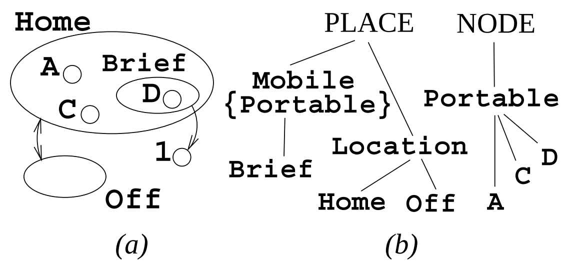

Figure 1. SetGraph encoding of Briefcase domain: (a) Graphical representation of $\alpha = \langle P_1, R_1 \rangle$; ($b$) Sort hierarchies

A *typed setGraph* is a setGraph in which some of the instances or R-values have been replaced with types or *variables* (having appropriate domain). We will refer to the label or variable name associated to a nodeSet as to that nodeSet's *identifier*. An *instantiated setGraph* is a typed setGraph in which all elements are associated to either instances, R-values, or numeric variables. A *ground setGraph* is an instantiated setGraph containing no variables (e.g., see Fig. 1.(a)).

2.2 Extending the Action Representation

In addition to the description of the initial world state, which is encoded as a *ground* setGraph, a planner must also be provided with a (declarative) specification of how states are changed by actions.

The original setGraph model [5] was limited to actions consisting only of nodeSet movement. Here, we consider the following possible state (setGraph) transformations: (α) *addition* or *removal* of an element, (β) *movement* of a nodeSet, and (δ) *re-assignment* (or *update*) of an R-value associated to one of the nodes. The movement (or 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. Given the current R-value x of a node and $v \in \Re_\perp$, the possible update operations are: (a) assignment ($x := v$); (b) increase ($x := x + v$); (c) decrease ($x := x - v$); (d) scale-up ($x := x \cdot v$), and (e) scale-down ($x := x/v$). Finally, any element not moved, removed or updated is left unaltered (i.e., we assume *default persistence*).

As usual, the domain-specific legal transformations of a state (ground setGraph) will be defined through a set of action schemata (operators). An operator $O \equiv (\Pi_B \Rightarrow \Pi_A)$ consists of preconditions Π_B (specifying the situation required to hold in the state *before* the action is executed) and effects Π_A (describing the situation of the state *after*). For example, Figure 2.(a) depicts a graphical representation of the "Move-briefcase" operator for the Briefcase domain, which allows moving a mobile object x (and its contents) from one location to another. The elements to the left of the arrow represent the preconditions; those to the right, the effects.

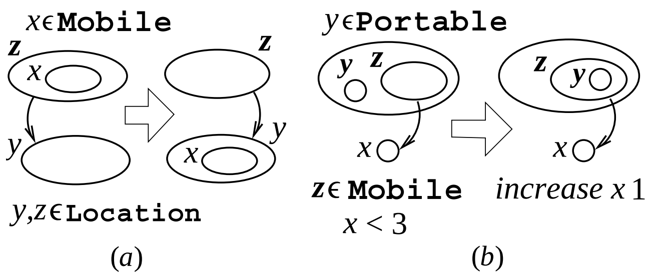

Figure 2. Briefcase domain: (a) "Move-briefcase" operator; (b) "Put-in" operator

Preconditions and effects are composed of two separate parts, analogical and numerical. The analogical component consists of an ordered list of *typed* setGraphs. The numerical part of the preconditions consists of a set of comparisons ($<, >, \leq, \geq, =, \neq$) between pairs of numerical expressions, while the numerical effects consist of a set of update operations of the kind (a)–(e) listed earlier. For example, Figure 2.(b) represents graphically the "Put-in" operator, which moves an object y inside a mobile z (subject to them being at the same location, and to the mobile containing at most two objects). The analogical precondition and effect lists of this operator consist of only one typed setGraph. The numerical parts constrain and update, respectively, the value $x \in \Re_\perp$ of the node associated to the mobile z.

The semantics of action are specified by providing an algorithmic definition of the following: (α) a method to check whether an operator O is applicable in a given state s; (β) a method for calculating the new state resulting from the application of an operator O to a state s. These definitions, given below, rely upon the following definition of *satisfaction*, which specifies the conditions for a ground setGraph G to 'match' a typed setGraph T. Intuitively, T and G match *iff* T can be made 'overlap' with G (or with parts of it) by replacing all of its sort labels and variables with instances and R-values, as appropriate:[1]

Definition 4 (Satisfaction) *Given three* NODE-PLACE-EDGE *type hierarchies, a typed setGraph T=*$\langle N, E \rangle$ *and a ground setGraph G,*

[1] The number of possible instantiated setGraphs that can be obtained from a typed setGraph by replacing sorts and sort variables with instances is finite.

T is satisfied *in G iff there is a substitution θ of all sort variables in T with instances of corresponding sorts, and a 1-1 function $\sigma : T \to G$ binding elements 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 i_T and i_G are the identifiers or R-values of x and $\sigma(x)$ after substitution θ, then either (1) $i_G = i_T$, or (2) i_G is an instance of i_T (i.e., $i_G \in i_T$), or (3) one is a numeric variable, and the other is an R-value or a numeric variable.*

For example, given the sort hierarchies of Figure 1.(b), the preconditions of the two operators of Figure 2 are both satisfied in the ground setGraph of Figure 1.(a). (E.g., for the preconditions of "Put-in" to be satisfied, either θ = (x/1, y/`A`, z/`Brief`) or θ' = (x/1, y/`D`, z/`Brief`) can be applied, with σ binding each of the labelled elements to the elements of Figure 1.(a) having identical identifier, the top place to place '`Home`' and edge (z,x) to edge (`Brief`,1)[2]).

We are now ready to define methods (α) and (β), presented earlier:

(α) An action schema $O \equiv (\Pi_B \Rightarrow \Pi_A)$ is *applicable* in a state (ground setGraph) s *iff* (1) all the typed setGraphs of Π_B are satisfied in s (through one substitution θ and one binding σ), and (2) all the numeric comparisons in Π_B are satisfied (after the appropriate variable/R-value replacements have been made).

(β) If operator O is applicable in state s, the *result* of applying O to s is the new ground setGraph obtained from s by (1) carrying out (on the corresponding elements of s identified through binding σ) the transformations required to change each of the setGraphs of Π_B into the (respective) final situation specified in Π_A, and (2) for each update operation of Π_A, updating the current R-values of the numeric nodes with the result of the given operations.[3]

3 THE SENTENTIAL MODEL: PDDL2.1-*lev*$_2^*$

The sentential representation adopted for this analysis is based on PDDL2.1 semantics [3], which builds on and extends the original core of Lifschitz' STRIPS semantics [11] to handle durative actions, numeric and conditional effectsn. The action description adopted here, however, is a simplified version of PDDL2.1, and is better thought of as extending STRIPS to numbers and functor symbols.

As in PDDL2.1 [3], the sentential world state is composed of two separate parts, a *logical* (STRIPS-like) state and a *numeric* state. While the logical state s is a set of atomic formulæ (the truth of an atom p depending on whether $p \in s$), the numeric state consists of a finite vector of R-values, containing all the current values of the possible *primitive numeric expressions* (PNEs) of the problem. A PNE is a formula $f(c_1, ..., c_n)$, where $c_i \in C$, set C contains all the objects of the domain, and f is a functor symbol such that $f : C^n \to \Re$ (Section 5 below provides a more precise definition).

A sentential operator $O \equiv (P \Rightarrow E)$ specifies a transformation of a state-pair s =(*logical*, *numeric*) into a new state-pair s'. In this simplified version, the preconditions P contain a set of (possibly negative) atoms and a set of comparisons between pairs of numeric expressions (containing PNEs and numbers). The effects E consist of a set of (possibly negative) atoms and a set of update operations of form "Op w *expr*", where 'Op' is one of the five update operators (a)–(e) used for the analogical action schemata (Section 2.2), w is a PNE, and '*expr*' is a numeric expression (combining PNEs and/or real numbers). The complete semantics for these operators is described in [3]. An example of sentential operator is given in the next section.

Finally, we note that any PDDL2.1 'level 2' (i.e., non-durative actions) operator can be compiled into an *equivalent set* of *ground* operators of the above form [3]. In view of this equivalence, we refer to the sentential formalism described above as to "PDDL2.1-*lev*$_2^*$".

4 THE HYBRID REPRESENTATION

The hybrid model 'glues' together the analogical and sentential models described above. In the hybrid representation, the world state is composed of two distinct parts: an *analogical* state and a *sentential* state. The two components are treated as two independent sub-states, much like logical and numerical states are treated separately in PDDL2.1. In particular, a hybrid operator consists of two distinct parts, each describing a transformation of the respective sub-state.

For example, consider a modified Briefcase domain, in which a bucket '`B`' containing green paint is used to carry around the portable objects `A,C,D`. Any object dropped in the bucket becomes green. The analogical part of the 'Drop-in' operator would be identical to the 'Put-in' action depicted in Figure 2.(b). The sentential part could consist of the following preconditions P and effects E:

P = { `(colour` y w`)` }
E = { `(colour` y `Green), (not (colour` y w`))`}

where $y \in$ `Portable` and w ∈`Colours`={`Green,Blue,`...}. As described in the previous section, the sentential part of the operator may also contain numerical elements. For example, given a 1-placed function '`Total_obj`' returning the number of items currently contained by a 'Mobile' object, precondition P could require `(< (Total_obj B) 3)`, and effect E would contain `(increase (Total_obj B) 1)`. This function is currently implemented in the analogical representation using a numeric node ('x' in Fig. 2.(b)).

5 SOUNDNESS OF HYBRID MODELS

The simple juxtaposition of sentential and analogical representations does not guarantee that the resulting model is *sound* with respect to the real domain represented. In this section, we describe a unifying framework that leads to the specification of the conditions for sound hybrid representations. These conditions extend those identified by Lifschitz in [11], which are restricted to sentential representations (and which are still at the basis of current planning languages [3]).

Following [11], the world is taken to be, at any instant of time, in a certain *state* s, one of a set S of possible ones. A domain consists of a finite set I of entities and finite sets of relations among (and properties of) entities. In order to describe a domain, we adopt a formal language $\mathcal{L} = \langle P, F, C \rangle$, where P, F, 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. The *wff* of $\mathcal{L}$, logical and numeric atoms, are built as follows:

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

[2] Any element of a typed setGraph having no specific label is considered as associated to the root label of the corresponding hierarchy.

[3] All the update operations are calculated using the 'old' R-values of the nodes, so that the order in which the updates are executed is irrelevant.

The symbols of $\mathcal{L}$ are given the standard semantics [2, Section 1.3]. In particular, an interpretation function g will map each constant symbol $c \in C$ to a distinct entity $i = g(c) \in I$, each m-placed *logical* function symbol $f \in F$ to a function $g(f) : I^m \to \Re$, and each n-placed *logical* relation symbol $p \in P$ to a relation $g(p) \subseteq I^n$. Each m-placed *numeric* function symbol $h \in F$ is mapped to a (fixed) function $g(h) : \Re^m \to \Re$, and each n-placed *numeric* relation symbol $q \in P$ to a (fixed) relation on real numbers $g(q) \subset \Re^n$. When $t \in \Re$, $g(t) = t$. If $t = f(t_1, ..., t_m)$, with $f \in F$ and $t_i \in C \cup$ PNE $\cup$ NE, then $g(t)$ is the value (in the current state s) of $g(f)$ calculated in $g(t_1), \ldots, g(t_m)$ (written '$f(t_1, \ldots, t_m) \mid_s$').

Definition 5 (Atom-satisfaction) *Given a language $\langle P, F, C\rangle$, a state $s \in S$ and an interpretation g, an atom $p(t_1, \ldots, t_n) \in \mathcal{L}$ is* satisfied *in s* iff *$g(t_1), \ldots, g(t_n)$ share relation $g(p)$ in s.*

In what follows we assume that, for a given language $\mathcal{L}$, a fixed interpretation 'g' is adopted.

Consider an abstract data structure $\mathcal{D}$ (such as a tree, a list, an array, etc.) and a universe $\mathcal{U}$ of elements (e.g., integers, characters, booleans,...). Let $\mathcal{D}_\mathcal{U}$ be a select set of *instances* of $\mathcal{D}$ built using elements in $\mathcal{U}$ (e.g., trees of booleans, of lists of integers, etc.).

Definition 6 (Model) *Given a language $\mathcal{L} = \langle P, F, C\rangle$ and a set $\mathcal{D}_\mathcal{U}$ of data structure instances with elements $\in \mathcal{U}$, a* model *is a pair M=(d, ϵ), where $d \in \mathcal{D}_\mathcal{U}$ and ϵ is a 1-1 total function $\epsilon : C \to \mathcal{U}$.*

A model is essentially a data structure containing elements taken from a set $\mathcal{U}$. The function ϵ maps the relevant objects (symbols) of the domain to the corresponding elements of the universe that represent them (which may or may not appear in the model). The use of an unspecified data structure $\mathcal{D}$ allows this definition to be used for both sentential (PDDL2.1-*lev*$_2^*$) and analogical (setGraph) models, as demonstrated, respectively, in Example 2 and Example 3 below.

Definition 7 (Domain representation structure) *A* domain representation structure *(DRS) for a language $\mathcal{L}$ with interpretation g is a triple $\langle \mathcal{D}_\mathcal{U}, \Psi, \Phi\rangle$, where $\mathcal{D}_\mathcal{U}$ is a set of instances of a data structure $\mathcal{D}$ with elements in $\mathcal{U}$, and each $\psi_i \in \Psi$ ($\phi_j \in \Phi$) is an algorithm associated to the n-placed (m-placed) relation (function) symbol $i \in P$ ($j \in F$), such that ψ_i, ϕ_j always terminate, and:*

- *for each logical relation symbol $p \in P$, $\psi_p : \mathcal{D}_\mathcal{U} \times \mathcal{U}^n \to \{0, 1\}$*
- *for each logical function symbol $f \in F$, $\phi_f : \mathcal{D}_\mathcal{U} \times \mathcal{U}^m \to \Re_\perp$*
- *for each numeric relation (function) symbol $q \in P$ ($h \in F$), ψ_q calculates $g(q) \subset \Re^n$ and ϕ_h calculates $g(h) : \Re^m \to \Re$*

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

Definition 8 (Model-representation) *Given a language $\mathcal{L}$, a DRS $\mathcal{R} = \langle \mathcal{D}_\mathcal{U}, \Psi, \Phi\rangle$ for $\mathcal{L}$ and a model M=(d, ϵ) with $d \in \mathcal{D}_\mathcal{U}$, M* represents *a state $s \in S$ (written 'M $\cong_\mathcal{R}$ s')* iff*, for every logical atom $p(t_1,...t_n)$ and PNE $f(t_1,...t_m)$ of $\mathcal{L}$, both of the following hold:*

- *$\psi_p(d, \epsilon(t_1), ..., \epsilon(t_n)) = 1$ iff $p(t_1, ..., t_n)$ is satisfied in s*
- *$\phi_f(d, \epsilon(t_1), ..., \epsilon(t_m)) = f(t_1, ..., t_m) \mid_s$ (if $f(t_1, ..., t_m) \mid_s$ is undefined, $\phi_f(d, \epsilon(t_1), ..., \epsilon(t_m)) = \perp$)*

Proposition 1 *Given a state $s \in S$, a DRS $\mathcal{R}$ for a language $\mathcal{L}$ and a model $M = (d, \epsilon)$ in $\mathcal{R}$, if $M \cong_\mathcal{R} s$ then, for all numeric atoms $q(t_1, ..., t_n)$ of $\mathcal{L}$,*

$$\psi_q(e(t_1), ..., e(t_n)) = True \textit{ iff } q(t_1,...t_n) \textit{ is satisfied in } s,$$

where 'e' is an evaluation function defined as follows:

- *if t is a number, $e(t) = t$*
- *if $t = f(t_1, ..., t_m) \in$ PNE, then $e(t) = \phi_f(d, \epsilon(t_1), ..., \epsilon(t_m))$*
- *if $t = f(t_1, ..., t_m) \in$ NE, then $e(t) = \phi_f(e(t_{j,1}), ..., e(t_{j,m}))$*

Example 2 - Consider the Briefcase domain. The briefcase, the two locations (home and office) and the three portable objects are the entities of interest. The property "*to be inside*" is the relation of interest, and the number of objects inside the briefcase is the only relevant numeric property. Let the language $\mathcal{L}_1$ contain the following symbols, having their standard interpretation: a 2-placed logical relation 'In', a 1-placed logical function 'Total_obj', and a 2-placed numeric relation '<'. The constant symbols are C_1={`A,C,D,Brief,Home,Off`}.

Let us build, for this domain and language, a *sentential* domain representation structure DRS$_1$ which replicates the semantics of PDDL2.1-*lev*$_2^*$. Accordingly, we choose to represent the state using a data structure $\mathcal{D}_1$ composed of a set W of strings and a numeric variable n with value $\in \Re_\perp$. The universe $\mathcal{U}$ consists of set C_1. Procedure $\psi_{In}(d, x, y)$ takes as input a data structure $d = (W, n)$ and two strings $x, y \in \mathcal{U}$, and returns '1' iff the string "In (x,y)" $\in W$ (after replacing x, y as appropriate). Procedure $\phi_{Total\_obj}(d, x)$ takes as input a structure $d = (W, n)$ and a string $x \in \mathcal{U}$, and returns the value of variable n if x="`Brief`", $\perp$ otherwise. Procedure $\psi_<(x, y)$ returns '*True*' iff x is smaller than y. Then, given a model M=(d, ϵ) such that $\epsilon : C_1 \to \mathcal{U}$ maps constant symbols to equivalent strings, M represents a Briefcase-domain state s *iff* the set W contains all and only the logical atoms of $\mathcal{L}_1$ which are satisfied in s, and variable n contains the current number of objects in the briefcase (i.e., the value of the PNE '`Total_obj`'). This encoding is essentially equivalent to the PDDL2.1 sentential model (see Section 3).

Notice that, thanks to Proposition 1, if M represents state s, procedure $\psi_<$ can be used to determine whether any (arbitrarily complex) numeric atom of $\mathcal{L}_1$ is satisfied in s – for example, whether <(`Total_obj`, 3) is satisfied.

Definition 9 (Planning Domain) *A* planning domain *is a pair $\langle S, A\rangle$, where S is the set of possible world states, and A, the set of actions, is a finite set of total functions $a : S \to S$.*

Given a domain $\langle S, A\rangle$ (with language $\mathcal{L}$ and DRS $\mathcal{R}$) and a set Σ of models in $\mathcal{R}$, Σ *represents* S *iff* each model $M \in \Sigma$ represents exactly one state $s \in S$, and $\forall s \in S, \exists! M \in \Sigma$ such that $M \cong_\mathcal{R} s$.

Definition 10 (Sound action model) *Given a domain $\langle S, A\rangle$ (with language $\mathcal{L}$ and DRS $\mathcal{R}$) and a set Σ of models in $\mathcal{R}$ representing S, a function $\lambda : \Sigma \to \Sigma$ is a* sound model *of $a \in A$* iff*, for each model $M \in \Sigma$ and state $s \in S$ such that $M \cong_\mathcal{R} s$, $\lambda(M) \cong_\mathcal{R} a(s)$.*

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

Theorem 1 (Soundness) *Let R=$\langle \Sigma, \Lambda\rangle$ be a sound representation of a domain D=$\langle S, A\rangle$. Let $\vec{\lambda} = \langle \lambda_1, ..., \lambda_n\rangle$ be a sequence (plan) of sound action models, and $\vec{a} = \langle a_1, ..., a_n\rangle$ be the corresponding sequence of actions. If $M_0 \in \Sigma$ represents $s_0 \in S$, and the application of $\vec{\lambda}$ to M_0 produces $M_n = \vec{\lambda}(M_0)$, then M_n represents $\vec{a}(s_0)$.*

Theorem 1 and the definition of sound action model extend to hybrid representations the 'Soundness Theorem' and 'Definition A' given in [11], which were restricted to purely sentential models (the proof is by induction and is analogous to the original proof [11]). In essence, the concept of *satisfaction* is replaced here with that of *representation*, applicable to both the sentential and analogical case.

The second result, presented below, shows that the analogical and sentential formalisms considered have *equivalent* expressive power:

Theorem 2 (Equivalence) *Any setGraph encoding of a planning domain can be transformed into an equivalent sentential (*PDDL2.1-*lev*$_2^*$*) description, and vice versa.*

A sketch of the proof is reported in the Appendix. The next example completes the setGraph encoding of the Briefcase domain. Notice that the equivalent sentential version, given in Example 2, is defined using the same theoretical framework (which is the object of this section).

Example 3 - Consider the Briefcase domain, with language $\mathcal{L}_1$ and interpretation as specified in Example 2. Let us define, for this domain and language, an *analogical* domain representation structure DRS$_2$. The data structure $\mathcal{D}_2$ adopted is the setGraph (an example of state was given in Figure 1.(a)). The universe $\mathcal{U}$ consists of set C_1.

Procedure $\psi_{In}(d, x, y)$ takes as input a ground setGraph d and two labels $x, y \in \mathcal{U}$, and returns '1' iff the setGraph of Figure 3.(a) (having parameters x, y substituted with the corresponding input) is *satisfied* in d. Procedure $\phi_{Total\_obj}(d, x)$ takes as input a ground setGraph d and a string $x \in \texttt{Mobile} \subset \mathcal{U}$, and, if $\exists\sigma$ such that the setGraph of Figure 3.(b) is satisfied in d with mapping σ, it returns the R-value $\sigma(w)$, $\perp$ otherwise. Procedure $\psi_<(x, y)$ works as usual.

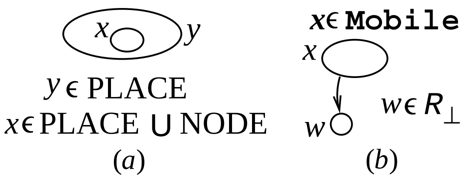

Figure 3. Briefcase domain: (a) typed setGraph encoding procedure $\psi_{In}(x, y)$; (b) typed setGraph encoding procedure $\phi_{Total\_obj}(x)$

6 RELATED WORK AND DISCUSSION

The main contribution of this paper is a sound theoretical framework (Definitions 5-10 and Theorem 1) for planning with analogical, sentential and hybrid representations. It should be pointed out that the framework described is *not specific* to the analogical or sentential models that have been considered here: although we have shown how both setGraphs and PDDL2.1-*lev*$_2^*$ representations can be supported, the theoretical model provides a basis for the integration of any other sentential and homomorphic representations that satisfy its premises.

The second contribution consists of an expressive analogical planning representation (Definitions 1-4), which extends the original 'setGraph' model [5] and makes it expressively equivalent to the sentential model adopted (Theorem 2). It should be noticed that although the extended formalism can ultimately encode any PDDL2.1 'level-2' planning domain description, it can do so only if conditional effects, quantification and disjunctive preconditions are previously compiled away [3]. In this sense, the expressiveness of the setGraph formalism is still limited. Nevertheless, we expect that adding this kind of 'syntactic sugar' to the model should not be too difficult.

A specific syntax for setGraph-based planning languages has not been discussed here. The BNF specification of a language for purely analogical planning was proposed by Garagnani and Ding in their original paper [5], but was restricted to the use of two-dimensional arrays of characters. While the full setGraph representation certainly requires a more complex definition, the simplicity of the elements upon which the model is built – lists, sets, nodes and pointers – should make a syntax specification relatively straightforward.

Myers and Konolige [15] described a hybrid framework for problem solving that allowed a sentential system to carry out deductive reasoning with and about diagrams. Their system allowed the addition (and extraction) of information to and from diagram models, but did not permit existing analogical information to be "retracted" from the models. This possibility is crucial for enabling non-monotonic changes of a diagram, typically associated with the execution of an action. Similar considerations apply to other works on heterogeneous representations, such as [1, 18]. The work of Glasgow and Malton on purely analogical, model-based spatial reasoning [7] is closely related to many of the ideas developed here; the present work generalises to hybrid models their approach, and extends it with a representation for describing and reasoning *about* the effects of an action.

The work of Long and Fox on the automatic detection of generic types [12] and on their use in problem decomposition [4, 13] is also relevant in this context. In particular, hybrid models can encode different aspects of a domain using distinct representations. For example, while analogical models can be used to encode spatial (or abstract) movement in a simple and efficient way, sentential representations appear to be more indicated for describing 'static' state changes. This ability to separate and solve independently the different components of a domain naturally leads to an effective problem decomposition. If the dynamics of a domain can be recast in terms of movement or manipulation of (abstract or physical) entities, the contraints that regulate the movement of such entities can be made *implicit* in an analogical (or hybrid) domain description, and significantly speed up the planning process.

Two important aspects of action modelling that have not been dealt with here concern the specification of *concurrent* and *durative* (analogical) actions. The conditions guaranteeing the non-interference of two sentential (PDDL2.1) operators and the semantics of sentential durative actions are discussed by Fox and Long in [3]. 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 acted upon by the operators be disjoint. However, the introduction of time and durative analogical actions, possibly in presence of other features – such as conditional and continuous effects – makes this a rather complex issue, requiring further investigation.

In summary, the ability to integrate different formalisms and combine their complementary and competing strenghts (such as efficiency, expressiveness and simplicity of encoding) allows planning language designers to develop models that overcome the weaknesses of purely sentential or diagrammatic representations. This paper lays the theoretical foundations for the development of sound, hybrid planning systems that integrate propositional and analogical models based on setGraphs or other, more advanced, homomorphic structures. The planner prototype and experimental results described in [6, 5] demonstrate the feasibility of the approach and the potential speed up of (purely) analogical planning. Thus, having addressed the basic practical and theoretical issues involved in the proposal, this work is now ripe for a full implementation and fielded application.

ACKNOWLEDGEMENTS

This work was partially supported by the UK EPSRC (grant no. GR/R53432/01). Thanks also to Michael Jackson, Neil Smith and Yucheng Ding for their useful comments and feedback.

REFERENCES

[1] J. Barwise and J. Etchemendy, 'Heterogeneous logic', in *[8]*, chapter 7, 211–234, (1995).

[2] C. Chang and H. Keisler, *Model Theory*, Elsevier Press, New York, 1977.

[3] 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).

[4] M. Fox and D.P. Long, 'STAN4: A Hybrid Planning Strategy Based on Sub-problem Abstraction', *AI Magazine*, **22**(4), (2001).

[5] M. Garagnani, 'Model-Based Planning in Physical domains using SetGraphs', in *Research and Development in Intelligent Systems XX: Proceedings of AI2003*, eds., M. Bramer, A. Preece, and F. Coenen, pp. 295–308, London, England, (December 2003). Springer-Verlag.

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

[7] J. Glasgow and A. Malton, 'A semantics for model-based spatial reasoning', Technical Report 94-360, Department of Computing and Information Science, Queen's University, Kingston, Ontario, (1994).

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

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

[10] J.H. Larkin and H.A. Simon, 'Why a diagram is (sometimes) worth ten thousands words', *Cognitive Science*, **11**, 65–99, (1987).

[11] V. Lifschitz, 'On the semantics of STRIPS', in *Proceedigns of 1986 Workshop: Reasoning about Actions and Plans*, eds., M.P. Georgeff and Lansky, (1986).

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

[13] D. Long and M. Fox, 'Extracting route-planning: First steps in automatic problem decomposition', in *Proceedings of AIPS'00 Workshop on Analysing and Exploiting Domain Knowledge for efficient Planning*, (2000).

[14] 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/).

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

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

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

[18] N. Swoboda and G. Allwein, 'A case study of the design and implementation of heterogeneous reasoning systems', in *Logical and Computational Aspects of Model-Based Reasoning*, eds., L. Magnani, N.J. Nersessian, and C. Pizzi, chapter 9, 3–20, Kluwer, Dordrecht, NL, (2002).

Appendix A

Theorem 2 (Equivalence) Any setGraph encoding of a planning domain can be transformed into an equivalent sentential (PDDL2.1-*lev*$_2^*$) description, and vice versa.

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

By definition, a setGraph is a pair $\langle N, E\rangle$, where N is a set of nodeSets and E a set of binary relations on $\wp(N)$. This structure can be easily encoded using two predicates (e.g., 'link(e, x, y)' and 'in(x, y)') expressing, respectively, the presence of edge (x, y) and that nodeSet x is an element of y. In addition, let each R-value r present in N be replaced by a label, uniquely identifying that node. The label can then be used as 0-placed function symbol and assigned the value r through the PNE vector of the numeric part of the sentential state. Given this encoding, every analogical transformation of a setGraph G into G' can be 'simulated' in the sentential representation by adding or removing the appropriate atoms to/from the current logical state L, so that L' represents G'. The numeric update of an R-value associated to a node is transformed into an update of the corresponding PNE in R.

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

Every state-pair $s = (L, R)$ consists of a finite set L of ground atoms p_i and a finite vector R of R-values, corresponding to the values y_j of the PNEs $\hat{f}_j$ in s. Let G be a ground setGraph that contains the following elements: (1) a place labelled "*True*"; (2) a distinct node (labelled "p_i") for each possible ground atom p_i of the domain language $\mathcal{L}$; and (3) a pair of nodes (having identifier "f_j" and R-value y_j, respectively) and one edge (linking f_j to y_j) for each PNE $\hat{f}_j$. Each node corresponding to an atom $p_i \in L$ will be placed inside *True*, while all the remaining nodes outside. Then, the truth of any atom p_i can be determined by checking if the setGraph (*True*$\{p_i\}$,$\emptyset$) is satisfied in G. In addition, the value of the PNE $\hat{f}_j$ is identified by the value of node $\sigma(w)$ to which the node labelled w has to be bound for typed setGraph ($\{f_j, w\}$,$\{(f_j, w)\}$) to be satisfied in G.

Given the above encoding, every setGraph operator can be transformed into an equivalent sentential operator as follows: each addition (removal) of a ground atom p_i to (from) the logical state L corresponds to the *movement* of node "p_i" inside (outside) place *True*. Similarly, each update of a PNE in R is encoded through the update of the R-value of the node in the corresponding linked pair.