In Bramer, M., Coene, F. & Allen, T. (Eds) Research and Development in Intelligent Systems XXI (Proc. of AI-2004), pp.214-227. Springer-Verlag, 2005

A Framework for Hybrid Planning

Max Garagnani
Department of Computing, The Open University
Milton Keynes - UK

Abstract

Sentential and analogical representations constitute two complementary formalisms for describing problems and domains. Experimental evidence indicates that different domain types can have their most efficient encoding in different representations. While real-world problems typically involve a *combination* of different types of domains, all modern planning domain description languages are purely sentential. This paper proposes a framework for planning with *hybrid* models, in which sentential and analogical descriptions can be integrated and used interchangeably, thereby allowing a more efficient description of realistically complex planning problems.

1 Introduction

Research in knowledge representation and reasoning [1, 2] indicates that many problems are easier to solve if described using *analogical* [3] (a.k.a. diagrammatic, homomorphic [4]) representations. Recent experimental evidence in planning [5] demonstrates that *Move* problems, involving the movement and manipulation of entities subject to a set of constraints, are solved significantly faster (up to two orders of magnitude) if recast in purely analogical terms. Real domains, however, are typically the result of a "mixture" of diverse types of components: representing all of these aspects in an efficient manner using a purely analogical (or a purely sentential) language is often impossible.

This work concerns the development of hybrid (or *heterogeneous* [4]) planning representations, able to merge sentential and analogical models into a single formalism that combines the strengths and overcomes the weaknesses of both paradigms. In this paper, we (*i*) review the *setGraph* model described in [6] (to the best of our knowledge, the only existing proposal of analogical planning representation) and extend it into a more expressive representation; (*ii*) briefly describe the sentential model chosen, based on the planning domain description language PDDL2.1 [7] and expressively equivalent to the analogical model; (*iii*) describe a simple model of hybrid planning which allows the two above representations to be integrated; and (*iv*) present a general theory that guarantees the soundness of the approach. In particular, the Soundness Theorem presented extends to analogical and hybrid representations the theory of *sound* action description given in [8], originally limited to sentential models. The final section discusses related work, limitations and future directions.

2 The Analogical Model: setGraphs

This section reviews the setGraph model proposed in [6] and extends it so as 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.

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 blocks and a table, represented by symbols A, B, C, Table. The vertices of the graph are depicted as ovals, labelled $\mathrm{V}_1, \ldots, \mathrm{V}_{10}$. 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 $\texttt{On}(x, y)$ holds in the current state.

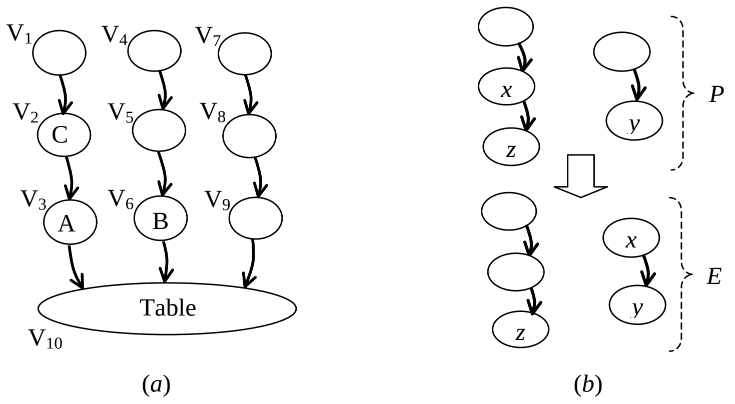


Figure 1: A SetGraph encoding of the Blocks World domain: (a) state representation; (b) *Move*(x, y, z) operator ($x \in \{$A,B,C$\}$; $y, z \in \{$A,B,C,Table$\}$).

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 re-arranged 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 is made more expressive by allowing edges to connect *any* two elements of the graph (e.g., a symbol and a vertex, or two symbols), and symbols to consist of strings of digits. Such an extended representation is presented more formally in the sequel.

2.1 Formalising SetGraphs

The formal definition of setGraph is based on the concept of *multiset* [9]. A multiset is a set-like object in which the order of the elements is unimportant, but their multiplicity is significant. For example, C={1,1,0,0,0} denotes a multiset of integers containing two occurrences of 1 and three occurrences of 0. Since the order is unimportant, C is equivalent to {1,0,1,0,0}, but not to {1,1,0,1,0}. The empty multiset is denoted as { }. We will say that x is *contained* in C and write "$x \in$ C" to indicate that element x appears (occurs) at least once in multiset C.

A setGraph is a multiset of *nodeSets*, which are defined as follows:

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

- *a symbol $w \in W$, or*
- *a finite multiset of 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. NodeSets that are symbols of W are called *nodes*. All other nodeSets are called *places*. Thus, a place can contain both nodes and places. Places can be labelled (with possibly identical labels). Given a nodeSet N, $\wp(N)$ is defined as the multiset of all the nodeSets occurring in N (including N itself). For example, consider a language W={A}. Let N_1 be the nodeSet {A, {A}, {{A}}}. Then, $\wp(N_1) = N_1 \cup$ {A, A, A, {A}, {{A}}, {A} }.

In order to represent numbers, the language W is allowed to contain also *numeric* symbols. A numeric node is a string of form $n.m$ or n (possibly preceded by $+$ or $-$), where n, m are sequences of digits and the first (last) digit of n (m) is not 0. The set of strings of all real numbers (of form "$n.m$") together with "$\perp$" (representing undefined values) will be called $\Re_\perp$.

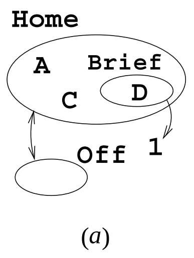
Definition 2 (setgraph) *A* setGraph *is a pair $\langle N, E\rangle$, where N is a nodeSet and $E = \{E_1, \ldots, E_k\}$ is a finite set of binary relations on $\wp(N)$.*

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 $\langle N_1, E_1\rangle$, with E_1={(C, B), ({B}, {{C}}), ({A,{B},{{C}}}, A)}, is a setGraph. The instances of the binary relation E_1, pairs of elements of $\wp(N_1)$, are the *edges* of the setGraph. Notice that if all places of N_1 were assigned distinct labels, for example, N_1=P_0{A, P_1{B}, P_2{P_3{C}}} (where the syntax *name*$\{x, y, ..., z\}$ denotes a place with label *name* containing nodeSets $x, y, ..., z$), then E_1 could be specified more simply as E_1={(C, B), (P_1, P_2), (P_0, A)}.

Example 1 - Consider the Briefcase domain, consisting of two connected locations (home and office), one briefcase, and three objects (called A, C and D). The sentential 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) }

Two (unlabelled) edges in R_1 are used to represent the bidirectional connection between the two locations. The setGraph also contains an edge associating place `Brief` to a numeric node. This node is used to keep track of the total number of objects currently inside the briefcase.



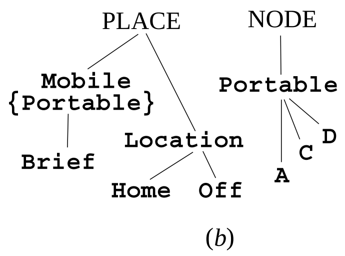


Figure 2: Analogical encoding of Briefcase domain: (a) graphical depiction of setGraph $\alpha = \langle P_1, R_1\rangle$ (see text); (b) nodeSets type hierarchy.

Figure 2.(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. NodeSets (and edges) of a setGraph are grouped in different types (or sorts), specified using a hierarchy (see Figure 2.(b)). By default, NODE and PLACE are the two only subtypes of the root-type NODESET (not shown in the figure). Every type t represents the finite set of *instances* (leaves) of the subtree that has t as root (hence, NODESET = PLACE ∪ NODE). Different types may have different properties. The properties of a type are inherited by all of its subtypes and instances. In this example, the PLACE hierarchy restricts any instance of a `Mobile` place to contain only `Portable` nodes (by default, a place may contain any instance of NODESET).

If a setGraph G has an associated type hierarchy, G is said to be *typed.* In a typed setGraph, the language W is assumed to contain all the instances of NODE $\cup \Re_{\perp}$. In what follows, all setGraphs will be assumed to be typed, unless otherwise specified. A *parameterised setGraph* is a setGraph in which at least one of the place labels or nodes has been replaced with one of its super-types (or, equivalently, with a variable ranging on one of its super-types). We refer to the label, symbol or variable name associated to an element x of a setGraph (nodeSet or edge) as to that element's *identifier*, returned by the function $id(x)$. A typed setGraph containing only instances of the NODESET hierarchy (i.e., no types or variables) is said to be *ground* (e.g., see Figure 2.(a)).

2.2 Representing Action

In addition to the description of the initial world state (encoded as a ground setGraph), a planner must also be provided with a specification of how states are changed by actions. As usual, the domain-specific legal transformations of a state are defined through a set of parameterised action schemata (operators). An operator $P \Rightarrow E$ consists of preconditions P, specifying the situation required to hold in the state *before* the action is executed, and effects E, describing the situation of the state *after.* For example, Figure 3.(a) depicts a graphical representation of the *Move* operator for the Briefcase domain, which transfers a mobile object x (and all of its contents) from one location to another. The elements to the left of the arrow represent the preconditions; those to the right, the effects.

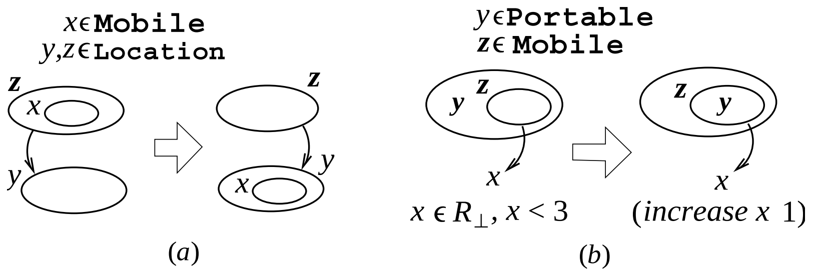


Figure 3: Briefcase operators: (a) *Move* and (b) *Put-in.*

While the model described in [6] was limited to actions consisting only of nodeSet movement, the addition and removal of an element and the update of the value of a numeric node will also be allowed here. The movement (or removal) of elements in a setGraph is based on the following general rules: (1) if a node is (re)moved, all edges linked to it move (are removed) with it; (2) if a place is (re)moved, all the elements contained in it and all edges linked to it move (are removed) with it. Given the current value x of a numeric node and a numeric value $v \in \Re_\perp$, 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*).

In a setGraph operator $P \Rightarrow E$, preconditions P and effects E are composed of two separate parts, analogical and numerical. The analogical components consist of 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 above. Numeric expressions are built from the values of numeric nodes using arithmetic operators. For example, Figure 3.(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 of the node associated to the mobile z.

The next definition specifies the conditions for a parameterised setGraph T to "match" (or be satisfied in) a ground setGraph G. Intuitively, T is satisfied in G *iff* there exists a substitution of all the parameters of T with appropriate instances such that T can be made "coincide" with G (or with part of it).

Definition 3 (Satisfaction) *Given a parameterised setGraph $T = \langle N, E \rangle$ and a ground setGraph G, T is* satisfied *in G iff there exists a substitution θ of all variables and types of T with corresponding instances, and a 1-1 function $\sigma : T \to G$ mapping elements of T to elements of G, such that, if T_θ is the ground setGraph T after substitution θ, the following conditions hold true:*

- *for each nodeSet (or edge) $x \in T_\theta$, $id(x) = id(\sigma(x))$*
- *for all pairs $(x, y) \in \wp(N) \times \wp(N)$, if $x \in y$ then $\sigma(x) \in \sigma(y)$*
- *for all edges $e = (x, y) \in E$, $\sigma(e) = (\sigma(x), \sigma(y))$*

The first condition requires that each element of T is mapped to an element having identical identifier. The second condition requires that any relation of containment between nodeSets of T is reflected by containment between the corresponding images in G. The last condition requires that if two nodeSets are linked by an edge, the corresponding images are linked by the image of the edge in G. For example, given the type hierarchy of Figure 2.(b), it is easy to see that the preconditions of the two operators of Figure 3 are both satisfied in the ground setGraph of Figure 2.(a).

The semantics of action is 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 in a state s. These definitions are given below:

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

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

3 The Sentential Model

The sentential planning representation adopted is a simplified version of PDDL-2.1 [7] equivalent to extending STRIPS to numbers and functor symbols.

As in PDDL2.1 [7], the sentential world state is composed here of two separate parts, a *logical* (STRIPS-like) state and a *numeric* state. While the logical state L consists of a set (conjunction) of ground atomic formulae (the truth of an atom p depending on whether $p \in L$), the numeric state consists of a finite vector R of values in $\Re_{\perp} = \Re \cup \{\perp\}$ (where $\Re$ is the set of real numbers). Each element of R contains the current value of one of the *primitive numeric expressions* (PNEs) of the problem (values associated with tuples of objects by functor symbols – see [7] for more details).

A sentential operator specifies a transformation of a state-pair $s = (L, R)$ into a new state-pair $s' = (L', R')$. We consider operators $P \Rightarrow E$ in which the preconditions P contain just a set (conjunction) of literals (possibly negative atoms) and a set of comparisons between pairs of numeric expressions (containing PNEs and numbers), while the effects E consist of a list of literals and a set of numeric update operations (analogous to those allowed in setGraph operators). This does not cause any loss of generality, as 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 [7]. In view of this, we refer to the sentential formalism described above as to PDDL2.1-lev_2^*. The complete semantics for these operators is described in [7]. An example of sentential operator is given in the next section.

4 The Hybrid Representation

The hybrid model puts 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; each hybrid operator will consist of two distinct parts, each describing the 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 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 3.(b). The sentential part could consist of the following preconditions P and effects E:

$P = \{$ `(colour` y w`)` $\}$
$E = \{$ `(colour` y `Green),` $\neg$`(colour` y w`)`$\}$

where $y \in$ `Portable` and $w \in$ `Colours`. As described in the previous section, the sentential part of the operator may also contain numerical elements. For example, given the 1-placed functor symbol "`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)`. Notice that the function (or PNE) `(Total_obj Brief)` is realised in the analogical representation of Figure 2.(a) by linking `Brief` to a numeric node.

5 Soundness of Hybrid Planning 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 [8], still at the basis of current sentential planning languages [7]. It should be pointed out that the contents of this section are mostly theoretical constructs which, practically speaking, are not strictly necessary for implementing a working hybrid planning system.

Following [8], 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*'s of $\mathcal{L}$ (logical and numeric atoms) can be built (only) as follows:

- $p\,(c_1, ..., c_n)$ is a *logical atom iff* $p \in P$ and $c_i \in C$ (with $i \in \{1, ..., n\}$)
- $q(t_1, ..., t_n)$ is a *numeric atom iff* $q \in P$ and $t_i \in$ PNE $\cup$ NE
- $f(c_1, ..., c_m)$ is a *primitive numeric expression* (PNE) *iff* $f \in F$ and $c_i \in C$
- a real number is a *numeric expression* (NE); $h(t_1, ..., t_m)$ is a NE *iff* $h \in F$ and $t_i \in$ PNE $\cup$ NE

where NE and PNE indicate, respectively, the sets of all numeric and primitive numeric expressions obtained only as described above.

An interpretation function g maps 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

$g(f) : I^m \rightarrow \Re$, 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 $g(h) : \Re^m \rightarrow \Re$, and each n-placed *numeric* relation symbol $q \in P$ to a (fixed) relation on real numbers $g(q) \subset \Re^n$.

We define $g(t) = t$ for all $t \in \Re$. Let $f \in F$ and $t_i \in C \cup$ PNE $\cup$ NE; if $t = f(t_1, ..., t_m)$, then $g(t)$ is the value (in the current state s) of $g(f)$ calculated in $g(t_1), \ldots, g(t_m)$ (written $f(t_1, \ldots, t_m)|_s$). In what follows, we assume that, for a given language $\mathcal{L}$, a fixed interpretation g is adopted.

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

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

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

Definition 6 (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 a procedure associated to the $n(m)$-placed relation (function) symbol $i \in P$ ($j \in F$), such that ψ_i, ϕ_j always terminate, and:*

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

Basically, a DRS consists of a data structure and a set of procedures for checking it. Each procedure 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 is sufficient to run the 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.

Example 2 - Consider the Briefcase domain of Example 1. 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`}.

For the above domain and language, let us define an *analogical* domain representation structure DRS$_a$. The data structure $\mathcal{D}_a$ adopted is the setGraph (an example of state was given in Figure 2.(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 if the setGraph of Figure 4.(a) (having parameters x,y substituted with the corresponding input) is *satisfied* in d, 0 otherwise. (Notice that $\psi_{In}(d,x,y)$ will need to check whether input label x is an instance of NODE or PLACE, and represent x as a place only in the latter case – this is indicated in the Figure using a dashed oval instead of a normal one). Procedure $\phi_{Total\_obj}(d,x)$ takes as input a ground setGraph d and a string $x \in$ `Mobile`, and, if there exists σ such that the setGraph of Figure 4.(b) is satisfied in d with mapping σ, it returns the value of $\sigma(w)$, $\perp$ otherwise. Procedure $\psi_<(x,y)$ calculates the binary relation "smaller than" on the set $\Re$ of real numbers.

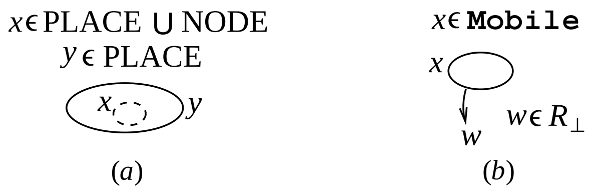


Figure 4: Briefcase domain: parameterised setGraphs encoding procedures $\psi_{In}(x,y)$ and $\phi_{Total\_obj}(x)$ ((a) and (b), respectively).

Definition 7 (Model-representation) *Given a language $\mathcal{L}$, a DRS $\mathcal{R} = \langle \mathcal{D}_\mathcal{U}, \Psi, \Phi \rangle$ for $\mathcal{L}$ and a model M=(d,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*
- $\phi_f(d,\epsilon(t_1),...,\epsilon(t_m)) = f(t_1,...,t_m)\,|_s$ *if $f(t_1,...,t_m)$ is defined in s, $\perp$ otherwise*

Definition 8 (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 $\forall s \in S$, there exists one and only one model $M \in \Sigma$ such that $M \cong_\mathcal{R} s$.

Definition 9 (Sound action representation) *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 representation *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 representations 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 representations, 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 representation extend to hybrid representations the Soundness Theorem and Definition A given in [8]. (The proof, by induction, follows directly from Definition 9, and is analogous to the proof given in [8], except that the concept of *satisfaction* is replaced here with that of *representation*, applicable to both the sentential and analogical case).

Finally, the following theorem demonstrates that setGraphs and PDDL2.1-*lev*$_2^*$ 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.*

Proof – The proof of the first part is uninteresting and not reported here for reasons of space. 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 show how any sentential operator can be encoded as an equivalent setGraph operator in this representation.

Every state $s = (L, R)$ consists of a finite set L of ground atoms $p(x_1, \ldots, x_n)$ and a finite vector R of numeric values y_j, each one representing the value in s of the j-th PNE $f(x_1, \ldots, x_m)$ (where $x_i \in C$ are constant symbols representing the entities of the domain). Let G be a setGraph containing the following: (1) three places, labelled *Pred*, *Obj* and *Funct*; (2) a node x_i in *Obj* for each symbol $c_i \in C$, with $id(x_i)$ =“c_i”; (3) a node with identifier “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 with identifier “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 in R (representing $f(x_1, \ldots, x_m)|_s$). Then, the truth of $p(x_1, \ldots, x_n)$ can be determined by checking if setGraph $\langle\{Pred\{p, x_1, \ldots, x_n\}\}, \{e_1(p, x_1), \ldots, e_n(p, x_n)\}\rangle$ is satisfied in G. Moreover, the value of the j-th PNE is identified by the value to which the numeric variable $w \in \Re_\perp$ is 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.

Given the above encoding, every setGraph operator can be transformed into an equivalent sentential 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 *expressive* equivalence of two formalisms does not imply equivalent *efficiency*: two formalisms that are equally powerful still produce different descriptions of the same problem, possibly requiring a different number of solution steps even when solved using the same search algorithm. Indeed, this was the set up for the experiments reported in [5]: in each of five different domains, and for each test problem, the same problem was solved much faster (between two and one hundred and sixty times) if recast in purely analogical terms.

6 Related Work and Discussison

Perhaps the main contribution of this paper consists of a theoretical framework (Definitions 5-10 and Theorem 1) for planning with analogical, sentential and hybrid representations. Notice that the framework described is not specific to the analogical or sentential models that have been considered here: in addition to being able to support and integrate setGraphs and PDDL2.1-lev_2^* representations, the theory provides a basis for the integration of *any* sentential and analogical representations that fit its premises.

A further contribution consists of an extended, powerful analogical planning representation (Definitions 1-3) based on setGraphs [6], expressively equivalent to the sentential model adopted (Theorem 2). It should be noticed that although the setGraph 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 [7]. In this sense, the expressiveness of the setGraph formalism is still limited, although adding this kind of "syntactic sugar" to the model should not be too difficult.

The possibility of separating the analogical part of a domain description from the sentential one suggests that hybrid representations may be very effective in the automatic extraction of heuristics [10]. In particular, a useful heuristic for a planning problem could be obtained by simply ignoring the sentential (or analogical) parts of the hybrid description, and solving the "relaxed" problem thus obtained. The learning of domain-specific control knowledge also appears to be facilitated by the adoption of analogical and hybrid descriptions. To see this, observe that the ability to learn from the solution of different problems in the same domain depends heavily on the capacity to recognise common "patterns" in different solutions. Identifying such patterns in sequences of propositional states is generally much harder then in analogical representations.

SetGraphs borrow ideas from various related works in the area of analogical representations [2, 1], set and graph theory [9] and semantic networks (in particular, conceptual graphs [11]). However, the use of diagrammatic representations in planning appears to have no precedents. Myers and Konolige [3] describe a hybrid framework for problem solving that allowed a sentential system to carry out deductive reasoning with and about diagrams. Their system allows 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 (e.g., [4]). The work of Glasgow and Malton [12] on purely analogical spatial reasoning 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 & Fox [13] on generic types and automatic problem decomposition nicely dovetails with the present approach. In particular, hybrid descriptions can be used to encode different generic types using different representations. For example, while the movement aspects of a domain can be conveniently encoded using setGraphs, stationary changes (e.g., a change in colour or size) may be better described using a sentential formalism. Once decomposed, these sub-problems can be solved independently, using more efficient, special-purpose algorithms.

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 in [5], though it was restricted to the use of two-dimensional arrays of characters. While the extended setGraph formalism requires a more complex definition, the simplicity of its primitive components – sets and graphs – should make a syntax specification relatively straightforward.

An important aspect of action modelling that has not been dealt with here concerns the specification of *concurrent* (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 acted upon by the operators be disjoint. However, the introduction of time and durative actions, and of other features such as conditional and continuous effects, makes this a rather complex issue, requiring further investigation.

Acknowledgements

This work was supported by the UK EPSRC (ref. GR/R53432/01). Special thanks go to Michael Jackson, Neil Smith, David Smith, Jörg Hoffmann, Bernhard Nebel, Maria Fox and Derek Long for their useful comments and feedback.

References

[1] Glasgow, J., Narayanan, N., Chandrasekaran, B., eds.: Diagrammatic Reasoning. AAAI Press/The MIT Press, Cambridge, MA (1995)

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

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

[4] Barwise, J., Etchemendy, J.: Heterogeneous logic. In: [1]. (1995) 211–234

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

[6] Garagnani, M.: Model-Based Planning in Physical domains using Set-Graphs. In Bramer, M., Preece, A., Coenen, F., eds.: Research and Development in Intelligent Systems XX (Proc. of AI-2003), Springer-Verlag (2003) 295–308

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

[8] Lifschitz, V.: On the semantics of STRIPS. In Georgeff, M., Lansky, eds.: Proceedigns of 1986 Workshop: Reasoning about Actions and Plans. (1986)

[9] Skiena, S.: Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Addison-Wesley, Reading, MA (1990)

[10] Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302

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

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

[13] Long, D., Fox, M.: 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 (AIPS'00), Breckenridge, CO, AAAI Press (2000) 196–205