# A Look at Function Symbols and Planning

Éric Jacopin *
Laforia, 4 place Jussieu, Boîte 169
France, 75252 Paris Cedex 05
jacopin@laforia.ibp.fr

**Abstract** This paper attempts to present how to use function symbols in action descriptions so that the planning algorithm only builds plans that are solutions to a planning problem. First, some coding tricks can be used so that action descriptions of the STRIPS framework can have their postconditions functionally dependent on the input situation. Since, in the STRIPS planning framework, a formula is true in a state if and only if it belongs to the set of formulas describing a state, using a function term in a STRIPS action description is impossible. Then, using ADL, it is shown how function terms can be used in action descriptions.

## 1 Introduction

**Assumptions** It is assumed that the planning framework addressed here has the following properties: (i) The environment is a set of discrete states, which can be represented by a set of formulae (i.e. as in STRIPS), but can also be the model of the set of formulas that represents the state (i.e. as in ADL ). (ii) Action descriptions characterize how one transforms a state into another one. In STRIPS, one uses the standard set operations (i.e. union and difference); and in ADL, one tranforms a model that interprets formulas into another one with rules defined in [5]. (iii) Providing it is correct, no particular algorithm is required to "plan" action descriptions.

**The motivation** of this paper relies in the following problem [1, page 350] (henceforth, I refer to this problem as the CEBW problem): "Consider a blocks world in which zero, one or two blocks can be on any given blocks. (This example and its analysis is due to David McAllester, personal communication.) Every block still must be on zero or one other block. We need a function `space` that takes a block as an argument and returns an integer between zero and two inclusive that tells how much room is left on top of the block. A precondition of `(puton a b)` is that `(space b)` be greater than zero, and the corresponding postcondition is that `(space b)` be one less than before. TWEAK can not represent this `puton`; a representation with conditional postconditions or postconditions that are functionally dependent on the input situation could. If TWEAK were extended to express this action, the modal truth criterion would fail."

Chapman gives an example illustrating why the MTC would fail: it is that of putting two blocks on top of another one. Then, the MTC allows the construction of plans that stack a third block although there can be at most two blocks on top of another.

In this example, one wish to introduce the

*Thanks to the anonymous referees for helpful remarks, to Jacques Ferber and his Group, and Philippe Morignot for their support in drafting this paper.

function term space. However, the TWEAK representation only deals with predicates. But as Lifschitz remarks it [4] (restriction on the definition A so any formula is an atomic formula), a sound STRIPS system may use functions as parameters of predicates. So why cannot TWEAK , which is a sound procedure, solve the CEBW problem? Moreover, how could one plan with terms only (i.e. terms not being parameters of predicates)?

Despite the general undecidability of planning with function symbols [1, 3], the use of function symbols in planning should be addressed in order to answer the previous questions.

# 2 Functions and the MTC

In this section, the reader must be familiar with TWEAK's basic terminology.

This section shows that functions terms can be used as parameters of predicates so that the CEBW problem can be solved by the MTC. I first present the language used in TWEAK (this is important since I need to extend it to solve the CEBW problem), then I extend the language with respect to Lifschitz's soundness [4] and finally discuss what TWEAK can't do.

## 2.1 TWEAK's language

TWEAK's alphabet only consists of predicate symbols, constant symbols, variables, the negation ¬ and the usual punctuation symbols "(" and ")". Terms are only made of variables or constants. So, the syntax is such that formulae are made of predicates that can be negated. The writing of formulae is based on LISP (on a is binary predicate and a and b are constant symbols): instead of the usual on(a,b) writing, the TWEAK writing is (on a b). Finally, (¬ (on a b)) is the negation of the previous formula.

The action description for the CEBW problem looks something like this (Chapman does not give it precisely):

| | |
|---|---|
| Name(puton) : | puton(x,y,z) |
| Pre(puton) : | {(on x z), (greater 0 (space y))} |
| Add(puton) : | {(on x y), (subtract (space y) 1)} |
| Del(puton) : | {(on x z)} |

Although it is not clear how the subtract symbol works (according to the language it could be a predicate), the key point is the greater predicate. This predicate tests if there is enough room on top of a block. For this to be true, (space y) must either return 1 or 2. Consider the partially ordered plan where, in parallel, three blocks are to be stacked on top of a fourth one. In the initial situation, the four blocks are on top of the table and in the final situation, three blocks are on top of the fourth one. This plan should be rejected since we consider that at most two blocks can be on any given blocks. But the MTC fails refusing the plan because stacking one block only subtract one to space. And then the precondition (greater 0 (space y)) of each of the three puton of the plan can be established. Consequently, as stated, Chapman concludes that [1, page 350]: "TWEAK can not represent this puton; a representation with conditional postconditions or postconditions that are functionally dependent on the input situation could."

## 2.2 Atomic Formulas

Let us assume TWEAK now deals with first order literals such as (greater 0 (space y)).

Lifschitz shows [4] that, if restricted to atomic formulae, a STRIPS system is sound: the application of a plan transforms the initial state into the desired final state. So why is TWEAK unsound?

The application of a plan is a linearization of the parallel plan that TWEAK refuses to reject. A totally ordered plan containing three puton descriptions should not be correct: if we apply the first two descriptions, then the space function evaluates to 0, the greater predicate is false and so is the precondition "(greater 0 0)" of the third description. But this supposes that the greater predicate has been interpreted so as to evaluate its truth. Unfortunately, there is no interpretation process during the TWEAK planning process: a formula is true in a state (if and) only if it belongs to the set describing this state. (greater 0 0) is false after the application of the first two puton descriptions if and only if (greater 0 0) does not belong to the resulting state. But the above puton neither asserts a greater predicate nor it deletes one.

Therefore, I propose to encode a puton description that simulates the adding and the deleting a greater predicate:

| | | |
|---|---|---|
| Name(puton) | : | puton(x,y,z,n,m) |
| Pre(puton) | : | {(on x z), (left x (space (space 0))), (left y (space n)), (left z m)} |
| Add(puton) | : | {(on x y), (left y n), (left z (space m))} |
| Del(puton) | : | {(on x z), (left y (space n)), (left z m)} |

This new puton description uses a binary predicate, left, which tells how much room is left on top of its first parameter. The space function is still needed, but is now iterated to simulate the increase and the decrease of the room left on top of a block. Finally, one block can be stacked on top of another if there is some room left: the precondition (left y (space n)) encodes this. Since there is no interpretation of formulae during the planning process, the iteration of the space symbol is visible in each set of formulae describing a state. (left y (space n)) thus tells that there must be at least one iteration of space for the description to be applicable. Then a STRIPS system is sound in using such a description and need not interpretation of formulae: TWEAK should not fail on refusing the partially ordered plan containing three new puton descriptions.

In the plan containing three parallel new puton descriptions, each of them clobbers the precondition (left y (space n)) since it belongs to their delete list. Then, ordering constraints must be added which finally entails the rejection of the plan.

The conclusion that TWEAK cannot represent the first puton is true but is also misleading: the planner does not interpret formulae at any stage of the planning process, but only tests the occurrence of a formula in a set. So TWEAK could not plan correctly with such a description. However, the CEBW problem can be solved with one function symbol, providing a correct encoding. But the trick of iterating the function cannot be done on any function symbol and thus one cannot fully escape the problem of interpreting formulae.

## 2.3 Function Terms

What is important in Chapman's puton is that it asks for the explicit interpretation of formulae: the subtract function term means that the space function should be updated to be one less than in the state where the description is applied.

A TWEAK state description is a set of possibly negated predicates. A predicate under an interpretation is either true or false whereas the interpretation of a function term returns a value according to the value of the parameters: a function term cannot be either true or false.

Consequently, planning with a function term as subtract is twofold: (i) it requires explicit interpretation of symbols and (ii) since this interpretation returns the value of function symbols, this treatment cannot be integrated to the general process of the formulas of a state where the values are the truth values; this treatment must update the value of the interpretation correspondingly with the values

| Formula of Update($\alpha$) | $\upsilon_{F_m}(y_1,...,y_m,w)$ |
|---|---|
| $F_m(t_1,...,t_m) \leftarrow t$ | $(y_1 = t_1 \wedge ... \wedge y_m = t_m \wedge w = t)$ |
| $F_m(t_1,...,t_m) \leftarrow t$ if $\psi$ | $(y_1 = t_1 \wedge ... \wedge y_m = t_m \wedge w = t \wedge \psi)$ |
| $F_m(t_1,...,t_m) \leftarrow t$ for all $z_1,...,z_k$ | $\exists z_1...z_k(y_1 = t_1 \wedge ... \wedge y_m = t_m \wedge w = t)$ |
| $F_m(t_1,...,t_m) \leftarrow t$ for all $z_1,...,z_k$ s.t. $\psi$ | $\exists z_1...z_k(y_1 = t_1 \wedge ... \wedge y_m = t_m \wedge w = t \wedge \psi)$ |
| If no formula concerns $F_m$ then $\upsilon_{F_m}(y_1,...,y_m,w) \equiv$ False | |

Table 1: Formulas corresponding to the "Update" field.

of the arguments of the function. ADL treats terms accordingly with these requirements.

# 3 Functions and ADL

This section assumes that the reader is familiar with structure-based first order logic [2].

This section first defines how ADL handles function terms and then show how the CEBW problem can be solved within ADL .

## 3.1 Definitions

**Definition 3.1 (Update field of an ADL action description)** *An ADL action description $\alpha$ is labeled with a syntactic expression of the form $\alpha(x_1,...,x_n)$ where $x_1,...,x_n$ are $n$ variables that parameterize $\alpha$. The Update($\alpha$) is a set of formulae of the form "$T \leftarrow V$" where $T$ and $V$ are (first order) terms. The "$\leftarrow$" connective indicates that the current interpretation of term $T$ is now updated to the interpretation of term $V$. The possible formulae of Update($\alpha$) are the following ($F_m$ is an $m$-ary function symbol, $t_1,...,t_m$ and $t$ are (first order) terms, $\psi$ is a (first order) formula and $z_i$ is a variable that appears in $t_1,...,t_m$ or $t$ but that does not appear in $x_1,...,x_n$):*

1. $F_m(t_1,...,t_m) \leftarrow t$.
2. $F_m(t_1,...,t_m) \leftarrow t$ *if* $\psi$.
3. $F_m(t_1,...,t_m) \leftarrow t$ *for all* $z_1...z_k$.
4. $F_m(t_1,...,t_m) \leftarrow t$ *for all* $z_1...z_k$ *such that* $\psi$.

ADL handles the "Update" field as follows:

**Definition 3.2 (ADL interpretation of a function symbol)** *Let $F_m$ a function symbol of arity $m$; $s$ is the current state and $r$ is the state resulting of the application of the ADL action description $\alpha$ to $s$. $r(F_m)$, the interpretation of the function symbol $F_m$ in state $r$ is as follows ($\mathcal{M}$ is the first order model of the formulas of state $s$):*

$$r(F_m) = \{(u_1,...,u_m,u_w) | \\ \mathcal{M} \models \upsilon_{F_m}(y_1,...,y_m,w) \vee \\ (\neg\exists(\upsilon_{F_m}(y_1,...,y_m,w)) \wedge F_m(y_1,...,y_m) = w)\}$$

*where $\upsilon_{F_m}(y_1,...,y_m,w)$ corresponds to a formula of Update($\alpha$) as explicated in Table 1 ($\equiv$ denotes equality between formulas).*

## 3.2 Revisiting CEBW

I here use only integers to solve the CEBW problem: blocks are encoded as integers ($\leadsto$ reads "is encoded to"): $Table \leadsto 3; A \leadsto 4; B \leadsto 5; C \leadsto 6; D \leadsto 7$. And space is as follows :

$$\mathsf{space}(x) = \begin{cases} 0 \text{ If } x = 0 \text{ ; } 1 \text{ If } x = 1; \\ 2 \text{ If } x = 2 \text{ or } x = 3; \\ \{0,1,2\} \text{ If } 7 \geq x \geq 4 \end{cases}$$

The binary predicate on now have integers as arguments. If $\mathsf{space}(block) = 2$, then *block* is clear; thus, the unary predicate *clear* is not needed. Consequently, the initial situation of the CEBW problem, where all the blocks are on the table, is defined by the following first order structure:

$$\mathcal{M}_{CEBW} = \langle\{0,1,2,3,4,5,6,7\}; \mathsf{on}; \mathsf{space}; 3\rangle$$
$$\mathcal{M}_{CEBW}(\mathsf{on}) = \{(4,3),(5,3),(6,3),(7,3)\}$$
$$\mathcal{M}_{CEBW}(\mathsf{space}) = \{(0,0),(1,1),(2,2),(3,2),(4,1),(5,0),(6,0),(7,0)\}$$

Note that the table is always clear; this is why 3, i.e. the integer encoding the table, is treated as a special constant in $\mathcal{M}_{CEBW}$. The subset {(0,0), (1,1), (2,2), (3,2)} always belongs to $\mathcal{M}_{CEBW}(\mathsf{space})$ whatever the blocks configuration is. The following puton action description encodes the problem in an ADL fashion:

| | |
|---|---|
| Name(puton) : | puton$(x,y)$ |
| Pre(puton) : | $\{x \neq y, x \neq Table, \mathsf{space}(x) = 2, \mathsf{space}(y) \neq 0\}$ |
| Add(puton) : | $\{\mathsf{on}(x,y)\}$ |
| Del(puton) : | $\{\mathsf{on}(x,z)$ for all $z$ such that $z \neq y\}$ |
| Update(puton) : | $\{\mathsf{space}(y) \leftarrow \mathsf{space}(y) - 1$ if $(y \neq Table)$ $\mathsf{space}(z) \leftarrow \mathsf{space}(z) + 1$ for all $z$ such that $(\mathsf{on}(x,z) \wedge z \neq y \wedge z \neq Table)\}$ |

where the precondition (greater 0 (space y)) is encoded into "$\mathsf{space}(y) \neq 0$".

# 4 Conclusions

The use of function symbols is possible in TWEAK as Lifschitz stated it. But the use of functions terms does not have any semantics in a planning framework where a formula is true in a state if and only if it belongs to the set describing this state. Again, this is exactly what Lifschitz stated in his paper.

However, one can use function terms if action descriptions change the way formulas are interpreted, and in particular, update the value returned by the function according to the values of the parameters. This is what ADL does.

This answers the question of wondering whether an action description with postconditions that are functionally dependent on the input situation could be used in TWEAK : one can tricky code the description so that the algorithm is correct but one does not escape the problem of updating the postconditions in function of the input situation. For this last point, one must use ADL.

# References


[1] D. Chapman, *Planning for Conjunctive Goals*, Artificial Intelligence **32** (1987), pp. 333–377.

[2] D. van Dalen, *Logic and Structure*, Universiteixt, Springer-Verlag ($3^{rd}$ printing, 1989), 207 pages.

[3] K. Erol, D. Nau & V. Subrahmanian, *Complexity, Decidability and Undecidability Results for Domain-Independent Planning*, Tech. Report, Computer Science Department, University of Maryland (1991), 46 pages.

[4] V. Lifschitz, *On the semantics of STRIPS*, in:*Reasoning about Actions & Plans*, Proceedings of the 1986 Workshop, Timberline, Oregon, Michael Georgeff & Amy Lansky Eds., Morgan Kaufmann (1987), pages 1–9.

[5] E. Pednault, *Towards a Mathematical Theory of Plan Synthesis*, Ph.D. Thesis, Electrical Engineering Department, Stanford University (December 1986), 245 pages.