Classical AI Planning as Theorem Proving: The Case of a Fragment of Linear Logic

Éric Jacopin[*]
Laforia-IBP, Boîte 169,
Université Pierre et Marie Curie,
4 place Jussieu,
F-75252 Paris Cedex 05
jacopin@laforia.ibp.fr

Abstract

This paper attempts to evaluate the use of a theorem prover in the multiplicative fragment of linear logic which has been shown to simulate conjunctive STRIPS-like planning [9]. A proof search procedure is presented that is correct, complete and only generates linear proofs (i.e. not trees). Plans that can be extracted from proofs are either totally or partially ordered. The procedure is tested against STRIPS-like planners and results are given. However, since linear logic is a resource-sensitive logic viewing formulas as data types, partial description of the final situation are impossible in linear logic; and shared postconditions are impossible in the fragment presented here. It is then argued that these restrictions eventually makes the presented fragment of linear logic, despite its formal framework, somewhat useless for practical planning purposes.

1 Introduction

Framework The linear logic framework is that of [9] and table 1 gives its related sequent calculus. The classical STRIPS planning framework is described in [3, 1]; plans are partially ordered sets of action descriptions. STRIPS action description is such that the performance of an action description only changes what is stated within the action description.

Motivation Despite some attempts to formalize the classical planning framework [1], conjunctive STRIPS-like planning still lacks a real logical framework. Recently, the concept of linearity has been shown to be useful for plan generation [4]. In refusing both weakening and contraction, linear logic [5] structurally handles this concept. It was then "logical" to attempt to introduce linear logic as a logic for plan generation. However, Masseron *et al.*'s formalization concentrated on expressing the adequacy between proofs and actions. Neither a proof search procedure nor a comparison with between classical planning expressiveness has been presented. This is the aim of this paper to both present a proof search procedure and to compare its efficiency and expressiveness with the classical STRIPS planning framework. It turns out that the procedure is inefficient compared to classical planners (for reasons due to the logic) and that expressiveness is poor in some cases.

Outline This paper is organized as follows. Section 2 briefly presents the fragment of linear logic discussed in this paper and its associated planning framework. A proof search procedure is given in section 3 and results from tests with STRIPS-like planners are given in section 4 as well as the requirements of a logic of action. The reader must have basic knowledge of both sequent calculus in linear logic [10] and classical planning [3, 1].

Identity and cut:

$$A \vdash A \qquad \frac{\Gamma \vdash A \quad \Gamma', A \vdash C}{\Gamma, \Gamma' \vdash C}$$

Mutliplicative Connective $\otimes$:

$$l\otimes\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \qquad \frac{\Gamma \vdash A \quad \Gamma' \vdash B}{\Gamma, \Gamma' \vdash A \otimes B}\ r\otimes$$

Table 1: The multiplicative linear sequent calculus to simulate classical conjunctive planning.

[*]Thanks to Marcel Masseron for discussions on Planning and Linear Logic; to Michel De Glas for discussions on the requirements for a logic of action; to Jacques Ferber and his group, MIRIAD, for their help in drafting this paper; and to Ivan Lamouret for remarks on previous drafts of this paper.

2 A small fragment of linear logic to simulate STRIPS planning

Readers are referred to [9] for a full description of the framework.

Formulas of the language are only atomic formulas or multiplicative of atomic formulas: $A_1 \otimes \ldots \otimes A_n$. The idea is to consider linear theories where one axiom describe one action. Thus a proof of a sequent in a theory uses action descriptions (i.e. sequents) and can thus be seen as a plan of actions. The sequent that is proved is the formal action and axioms that describes actions are concrete actions or action sequents. An action sequent is a sequent of the form

$$A_1, \ldots, A_m \vdash B_1 \otimes \ldots \otimes B_n$$

where each A_i and B_j are atomic formulas; if (Γ, Δ) denotes an action sequent then both Γ and Δ are multisets (i.e. sets allowing multiple occurences of an element). The formal action that must be proved has the same structure than an action sequent; its antecedent describes the initial situation of the planning problem and its succedent describes the final situation of the planning problem.

3 A proof search procedure

The idea of the procedure is the following: each leaf of a proof in a linear theory is either an identity axiom or an action sequent. An action has the same structure than the sequent that is to be proved: its antecedent is a multiset of atomic formulas and its succedent is a multiplicative formula. Consequently, one must try to apply an action sequent to the current situation (which is the antecedent of the current sequent to be proved) using a cut on the succedent of the action sequent and then eliminate the multiplicatives $\otimes$ with $l\otimes$. If there possibly exists some common atomic formula between the antecedent and the succedent of the sequent obtained from the cut, then one may eliminate them using $r\otimes$ with an identity axiom. This is entails the following formal definitions and results ($\sqsubseteq$ denotes the inclusion for multisets):

Definition 3.1 (Sub-sequent) *A sequent (u, t) is a sub-sequent of the sequent (s, r) if and only if $t \sqsubseteq r$.*

It is immediate that the binary relation "is a sub-sequent" is an order relation on the sub-sequents of a sequent.

Lemma 3.1 (Proof with a sub-sequent) *Let (A, C) be an action sequent applicable in a situation S; and let $R \neq S$ be another situation. There exists a proof of (S, R) in the linear theory composed of both (A, C) and (U, T) where (U, T) is a sub-sequent of (S, R).*

The proof makes use of the cut and then a systematic application of $l\otimes$ and then $r\otimes$ in a non-deterministic manner as in the previous informal discussion. One can derive CSLL (see below) from the precedent lemma and then prove this procedure correct and complete:

Theorem 3.1 (Correctness of CSLL) *Let P be a planning problem in linear logic. Any proof that CSLL constructs for P is a correct proof.*

Theorem 3.2 (Completeness of CSLL) *Let P be a planning problem in linear logic; and let (S, R) the formal action of P. If there exists a proof for P then there exists a sequence of application of the rules of the sequent calculus of table 1 that proves (S, R).*

CSLL has four parameters. The first is the set of axioms (i.e. action sequents of the theory; the second and third one are the antecedent and the succedent of the current formal action to be proved; and the last parameter contains the proof constructed by CSLL ($\sqcup$ denotes the union for multisets):

```
1   CSLL(Axioms,$E_i$,$E_f$,P)
2     **If** $E_i \vdash E_f$ is the identity **Then**
3       **return** P
4     **Else**
5       Choose(($A_{as}, C_{as}$),Axioms,$A_{as} \sqsubseteq E_i$)
6       **If** $(E_i = A_{as}) \wedge (E_f = C_{as})$ **Then**
7         **return** P
8       **Else**
9         $(A_{as}, C_{as}) = (\Gamma, B)$
10        $(E_i, E_f) = (\Gamma \sqcup \Gamma', C)$
11        P $\leftarrow$ P$\cup\{\frac{\Gamma \vdash B \quad \Gamma', B \vdash C}{\Gamma, \Gamma' \vdash C} Cut\}$
12        $(A_1, C_1) \leftarrow (\Gamma' \sqcup \{B\}, C)$
13        **While** $(A_1, C_1) = (\Gamma' \sqcup \{B' \otimes B''\}, C)$ **Do**
14          P $\leftarrow$ P$\cup\{l \otimes \frac{\Gamma', B', B'' \vdash C}{\Gamma', B' \otimes B'' \vdash C}\}$
15          $(A_1, C_1) \leftarrow (\Gamma' \sqcup \{B', B''\}, C)$
16        **End While**
17        $(A_2, C_2) \leftarrow (A_1, C_1)$
18        **While** $(A_2, C_2) = (\{C'\} \sqcup \Delta, C' \otimes C'')$ **Do**
19          Choose non deterministically:
20            1. P $\leftarrow$ P$\cup\{\frac{C' \vdash C' \quad \Delta \vdash C''}{C', \Delta \vdash C' \otimes C''} r\otimes\}$
21               $(A_2, C_2) \leftarrow (\Delta, C'')$
22            2. **Exit While**
23        **End While**
24        $(A_3, C_3) \leftarrow (A_2, C_2)$
25        CSLL(Axioms,$A_3$,$C_3$,P)
26      **End If**
27    **End if**
```

Where procedure Choose(e, S, C) choose an element e of set S with the constraints C on the choice of e. Thus, ligne 5 of CSLL chooses an action sequent in the set of possible action sequents such that the antecedent of the chosen action sequent is a subset of the current initial state.

The necessity to apply non-deterministically $r\otimes$ is illustrated by the proof of the two-register swapping of figure 1: one could apply $r\otimes$ with the identity axiom $cont(Z,A) \vdash cont(Z,A)$ instead of Cut2, but then *Assign(Z,Y,A,B)* could not be introduced.

A plan extraction procedure from a proof is given in [9]. This procedure tracks consumption and production of formulas through the application of the rules of the sequent calculus of the fragment. Note that the plans extracted from the proofs can be partially or totally ordered.

4 Discussion

This section first presents some results in comparing CSLL with the usual classical planning procedure and then discuss the notion of action within linear logic.

4.1 Testing

CSLL has been tested against a classical "forward chaining" planning procedure. Such a procedure tries to apply an action description in the current situation. This is done by checking that the preconditions of the action are included in the set describing the current situation. If this is the case, the resulting situation is computed from set union and set difference operations with, respectively, the add list and delete list of the action description. The action description is then added to the plan. It is obvious that CSLL works in the same spirit: first applying the cut on the succedent of an action sequent and then eliminating $\otimes$ on the left side of the resulting sequent is about the same as applying a STRIPS description to the current situation. Both procedures have been tested on the following examples (when a robot hand is used, all the action descriptions are such that all their preconditions are deleted):

1. **2R** is the two-register swapping problem from [9].
2. **e1** is the blocks world example from [9]; uses a robot hand. There exists a solution with one action description.
3. **e2** is like **e1** but there's only one bloc to manipulate. There exists a solution with four action descriptions.
4. **AS** is the so-called Sussman anomaly with the planning operators as in [1] (no robot hand). There exists a solution with two action descriptions.
5. **e3, e4, e5** are a progression toward **AS** but with a robot hand as in [9]: **e3** corresponds to put c on the table, b atop of c and the robot hand is empty; **e4** is **e3** plus the fact that the robot has to hold a; and **e5** corresponds to **AS**. **e5** has a solution with four action descriptions.

The test consists in counting the number of partial plans (i.e. plans that are not yet a solution) that have been generated by each procedure and then calculate the ratio of this number for CSLL over this number for the "forward chaining" planning procedure. The results are shown in figure 2. CSLL is rather inefficient when all the preconditions are deleted. This comes from the non-deterministic application of $r\otimes$: a lot of backtrack is due to it. However, up to now, it not clear how to transform this non-determinism into a systematic application or non application of $r\otimes$.

A second test has been done with the simulation of a Linear Bounded Automaton (LBA) which is a Turing machine whose tape is bounded to the length of the input[1]. Encoding from [2] has been used. Since the length of the tape is constant, a unique predicate encode the tape (i.e. one parameter corresponds to one tape square and two more parameters are needed to encode the position of the head and the current state of the machine). More than fifty operators are needed to encoded example 7.1 of [6]. All those operators are such that there is only one precondition and one postcondition; consequently, STRIPS encoding and action sequents are structurally equivalent: the behavior of the corresponding procedure should be comparable. Moreover, this test is interesting because despite the large number of possible action descriptions runtimes are not prohibitive. The test was conducted as follows. There is a solution involving seven action descriptions. Each procedure has been depth bounded and the total search space has been measured. Then when the given depth is, say, 5, there is no solution and the procedure must visit all its search space. On the figure, the black square and the white square represent CSLL and the "forward chaining" planning procedure respectively; and the black and white diamonds represent a "backward chaining" planning procedure (PWEAK [7], which is a minimal reconstruction of TWEAK [1]) and the LBA simulator, respectively. The figure speaks from itself: the precondition is not deleted (there is no such need since the parameter denoting the final state of the problem will make the predicate denote the unique final state of the machine) and then CSLL behaves like the classical "forward chaining" planning procedure. But PWEAK outperforms both procedures and behaves almost like the LBA simulator. This comes from the fact that STRIPS action descriptions have an empty delete list: the planner then needs not to check for clobberers of

[1]The acceptance problem for such a machine is PSPACE-Complete.

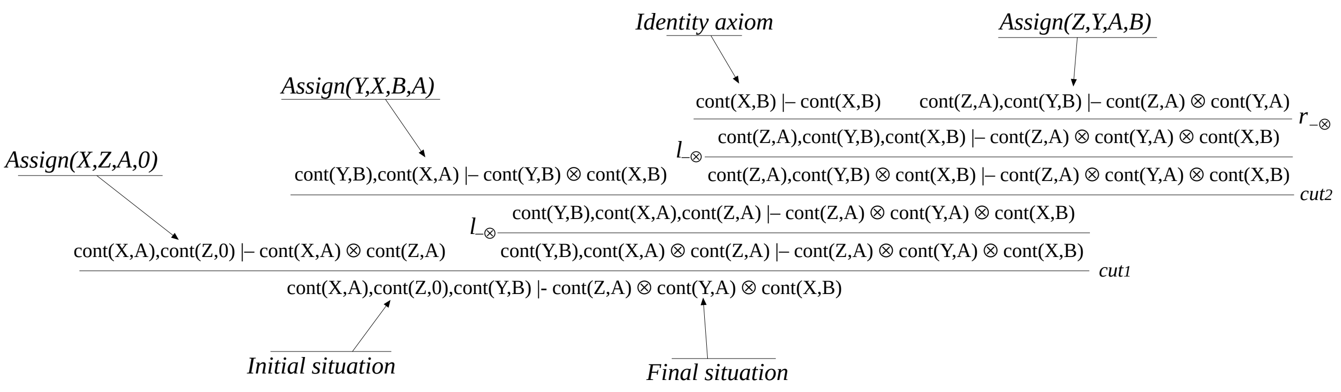
Identity axiom
Assign(Z,Y,A,B)
Assign(Y,X,B,A)
Assign(X,Z,A,0)
cont(X,B) |– cont(X,B)
cont(Z,A),cont(Y,B) |– cont(Z,A) ⊗ cont(Y,A)
r–⊗
cont(Z,A),cont(Y,B),cont(X,B) |– cont(Z,A) ⊗ cont(Y,A) ⊗ cont(X,B)
L⊗
cont(Y,B),cont(X,A) |– cont(Y,B) ⊗ cont(X,B)
cont(Z,A),cont(Y,B) ⊗ cont(X,B) |– cont(Z,A) ⊗ cont(Y,A) ⊗ cont(X,B)
cut2
cont(Y,B),cont(X,A),cont(Z,A) |– cont(Z,A) ⊗ cont(Y,A) ⊗ cont(X,B)
L⊗
cont(X,A),cont(Z,0) |– cont(X,A) ⊗ cont(Z,A)
cont(Y,B),cont(X,A) ⊗ cont(Z,A) |– cont(Z,A) ⊗ cont(Y,A) ⊗ cont(X,B)
cut1
cont(X,A),cont(Z,0),cont(Y,B) |- cont(Z,A) ⊗ cont(Y,A) ⊗ cont(X,B)
Initial situation
Final situation

Figure 1: A proof to swap two registers.

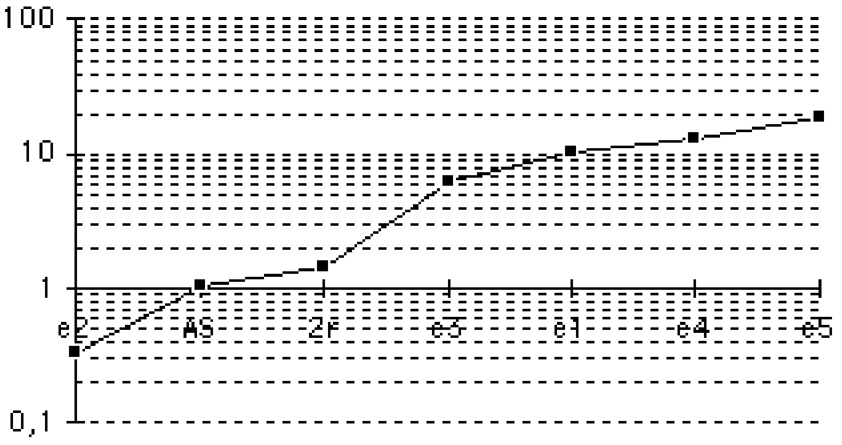

Figure 2: Proof search planning against "forward chaining" planning.

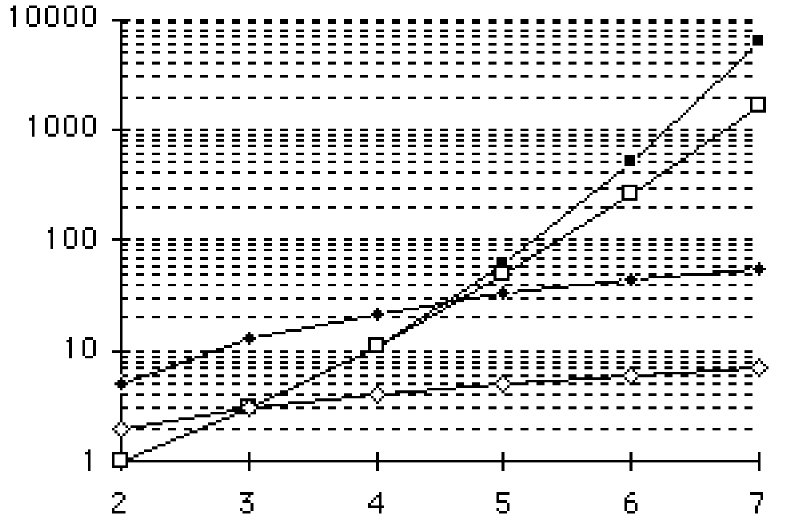

Figure 3: Total search space for a given depth to simulate a linear bounded automaton.

preconditions and thus only looks for possible establishers of the current goal. Since the state is a parameter of the predicate describing the LBA, the number of operators that can achieve a goal are limited to the number of operators that have the same state as parameter of the predicate describing the machine.

4.2 Partial final situations and sharing postconditions

But the main problem of CSLL is not inefficiency. Linear logic is a resource sensitive logic where formulas are the resource. Since the succedent of the formal action to prove is a multiplicative formula, the proof must then exactly produce this formula. To produce such a formula, the suppositions must be used only once. Consequently, there is no side effects on formulas: each produced formula must be used to prove the formal action. If this is not the case, then the proof is impossible. This has some terrible consequence within the linear logic framework presented. If an action is applicable in a situation and its effects do not exactly (formula by formula) match with the final situation, then the proof is impossible. This is illustrated with the two-register swapping example: one must say in the final situation which register will be stored in the extra register: one must say that X's value (i.e. $\underline{A}$) will be stored in $\underline{Z}$:

$$\begin{array}{l} cont(X, \underline{A}), cont(\underline{Z}, vide), cont(Y, B) \\ \vdash \\ \underline{cont(Z, A)} \otimes cont(Y, A) \otimes cont(X, B) \end{array}$$

This is clearly not tenable. This entails that one should say how many objects has been counted before counting them (note that this has lead the authors of [8] to encode a two counter machine such that the final situation is when both counters are zero).

Shared postconditions can be simulated with the storage operator of linear logic (i.e. "!") but this is merely a controlled contraction that one wanted to avoid in planning proofs (the linearity concept). Partial final situation is an unsolvable problem since this is a structural requirement of the logic. Finally, one should remark that, for the problem of formalizing action, it is the idempotency of action that should be controlled and not the idempotency of its preconditions and effects (through refusing weakening and contraction).

5 Conclusion

A proof search procedure for a small multiplicative fragment of linear logic is presented. The procedure is correct and complete and constructs linear proofs (i.e. not trees). The procedure has been tested against usual AI planners; and the procedure is rather inefficient when actions descriptions are such that the preconditions are all deleted. But the main problem relies in the expressiveness: neither partial descriptions of final state nor shared postconditions are possible in the proposed fragment. One must then enrich the fragment (maybe at the cost of undecidability [8]) so as to define a framework so that the idempotency of action is controlled whereas in linear logic, only the idempotency of pre- and postconditions of actions is controlled.

References

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

[2] K. Erol, D. Nau, et V. Subrahmanian, *Complexity, Decidability and Undecidability Results for Domain-Independent Planning*, Technical Report CS-2797, Computer Science Department, University of Maryland, 1991, 46 pages.

[3] R. Fikes, P. Hart & N. Nilsson, STRIPS*: A New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence **2** (1971), pages 198–208.

[4] B. Fronhöfer, *Linearity and Plan Generation*, New Generation Computing **5** (1987), pages 213–225.

[5] J.-Y. Girard, *Linear Logic*, Theoretical Computer Science **50** (1987), pages 1–102.

[6] J. Hopcroft & J. Ullman, *Introduction to Automata Theory, Language and Computation*, Addison-Wesley (1978), 418 pages.

[7] É. Jacopin & J.-F. Puget, *From* TWEAK *to* PWEAK*: Reducing the Non-Linear Planning Framework*, Laforia Technical Report 29/90, University of Paris 6, (December 1990), 11 pages.

[8] P. Lincoln, J. Mitchell, A. Scedrov & N. Shankar, *Decision Problems for Propositional Linear Logic*, Center for the Study of Language and Information, Report CSLI-91-147 (1991), 76 pages.

[9] M. Masseron, C. Tollu et J. Vauzeilles, *Generating Plans in Linear Logic: Actions as Proofs*, Theoretical Computer Science **113** (1993).

[10] A. Troelstra, *Lectures on linear logic*, Center for the Study of Language and Information, Lecture Notes **29** (1991) , 200 pages.