On Linear Logic Planning and Concurrency

Ozan Kahramanoğulları

Imperial College London, Department of Computing
ozank@doc.ic.ac.uk

Abstract. We present an approach to linear logic planning where an explicit correspondence between partial order plans and multiplicative exponential linear logic proofs is established. This is performed by extracting partial order plans from sound and complete encodings of planning problems in multiplicative exponential linear logic in a way that exhibits a non-interleaving behavioral concurrency semantics. Relying on this fact, we argue that this work is a crucial step for establishing a common language for concurrency and planning that will allow to carry techniques and methods between these two fields.

1 Introduction

Planning [1] and concurrency are two fields of computer science that evolved independently, aiming at solving tasks that are similar in nature but different in perspective: while planning formalisms focus on finding a plan, if there *exists* such a plan, that solves a given planning problem; the focus in concurrency theory is on the global behaviour of a given concurrent system, resulting in *universally quantified* queries, e.g., deadlock freeness, verification of a security protocol. In contrast to approaches to planning, in order to be able to handle such queries, languages for concurrency are equipped with a rich arsenal of mathematical methods that allow for an analysis of equivalence of processes.

In concurrency theory, parallel and sequential composition are expressed at the same level of representation, since they are equivalently important notions for expressing concurrent processes. However, in planning, although parallel behaviour between actions have been studied in partial order planners, e.g., UCPOP [28], Graphplan [1], these investigations focused on increasing the efficiency of the planners. In these approaches, the independence and causality between partially ordered actions, which is crucial from a concurrency theoretic point of view, is often specified by means of linguistic constraints (see, e.g., [19, 2]). Another line of research, which aims at capturing the concurrent behaviour of actions in the logical AI literature, e.g., in [30], defines concurrency over the parametrised time spans shared by the actions. [2]

Linear logic is widely recognised as a logic of concurrency (see, e.g., [26]) also because of its resource conscious features. In this paper, we propose linear

[1] A preliminary version of this paper has been presented as short paper at the 14th Int. Conference on Logic for Programming Artificial Intelligence and Reasoning.

[2] For a survey on reasoning about actions, planning and concurrency, see [13].

logic planning (see. e.g., [25, 24, 17, 18]) as a platform for a common language for planning and concurrency, aiming at bringing these two fields closer and allowing the techniques and tools in both fields to be interchanged. We establish a strict correspondence between partial order plans and the proofs of multiplicative exponential linear logic encodings of planning problems. The partial order plans which we extract from the proofs by an algorithm exhibit a non-interleaving behavioural concurrency semantics. Our result also contributes to the field of petri nets because of the strict correspondence between the reachability problem in petri nets and linear logic planning problems (see, e.g., [4]).

As the underlying formalism we employ the proof theoretic formalism of the calculus of structures (see, e.g., [11, 31]) instead of the sequent calculus. The distinguishing feature of this formalism is deep inference: the inference rules can be applied at arbitrary depths inside logical expressions. This brings about properties of proofs and deductive systems that are interesting from the point of view of computer science applications (see, e.g., [13]): in particular, in the complementary work in [14], the logic that we use heavily relies on a notion of deep inference as in the calculus of structures [11, 16], and it cannot be given as a sequent calculus system, as it was shown by Tiu [33]. Furthermore, more possibilities in the permutability of the inference rules in the calculus of structures yield optimised presentations (decomposition) of proofs [31]. By using the formalism of calculus of structures instead of the sequent calculus, it becomes possible to profit from these properties and also combine the results of this paper and the results of [14] for a common language for planning and concurrency, e.g., in [15]. Space restrictions do not allow to give the proofs here, we refer to [13].

2 Linear Logic Planning and Concurrency

In the following, we review multiplicative exponential linear logic in its deep inference presentation, following [31].

2.1 MELL in the Calculus of Structures

There are countably many *atoms*, denoted by $a, b, c, \ldots$ The *formulae* P, Q, R, $S \ldots$ of multiplicative exponential linear logic are generated by

$$R ::= a \mid \mathbf{1} \mid \bot \mid (\, R \parr R \,) \mid (\, R \otimes R \,) \mid !R \mid ?R \mid \bar{R} \quad ,$$

where a stands for any atom, $\mathbf{1}$ and $\bot$, called *one* and *bottom*. A formula $(R_1 \parr R_h)$ is a *par formula*, $(R_1 \otimes R_h)$ is a *times formula*, $!R$ is an *of-course formula*, and $?R$ is a *why-not formula*; $\bar{R}$ is the *negation* of the formula R. Formulae are considered to be equivalent modulo the relation $\approx$, which is the smallest congruence relation induced by the equations for associativity and commutativity for par and times formulae together with the following equations.

$$\begin{array}{llllll} (\bot \parr R) \approx R & ??R \approx ?R & !!R \approx !R & \overline{(R \parr T)} \approx (\bar{R} \otimes \bar{T}) & \overline{?R} \approx !\bar{R} & \bar{\bar{R}} \approx R \\ (\mathbf{1} \otimes R) \approx R & \bot \approx ?\bot & \mathbf{1} \approx !\mathbf{1} & \overline{(R \otimes T)} \approx (\bar{R} \parr \bar{T}) & \overline{!R} \approx ?\bar{R} & \end{array}$$

A *formula context*, denoted as in $S\{\ \}$, is a formula with a hole that does not appear in the scope of negation. The formula R is a *subformula* of $S\{R\}$ and $S\{\ \}$ is its *context*. Context braces are omitted if no ambiguity is possible.

An *inference rule* is a scheme of the kind $\rho\,\dfrac{T}{R}$, where ρ is the *name* of the rule, T is its *premise* and R is its *conclusion*. A typical deep inference rule has the shape $\rho\,\dfrac{S\{T\}}{S\{R\}}$ and specifies a step of rewriting, by the implication $T \Rightarrow R$ inside a generic context $S\{\ \}$, which is linear implication in our case. Rules with empty contexts correspond to the case of the sequent calculus.

The following rules give the multiplicative exponential linear logic system in the calculus of structures [31], or system ELS. The rules of system ELS are called *atomic interaction* ($\mathsf{ai}{\downarrow}$), *switch* (s), *promotion* ($\mathsf{p}{\downarrow}$), *weakening* ($\mathsf{w}{\downarrow}$), and *absorption* ($\mathsf{b}{\downarrow}$), respectively.

$$\mathsf{ai}{\downarrow}\,\frac{S\{1\}}{S(a \parr \bar{a})} \qquad \mathsf{s}\,\frac{S((R \parr T) \otimes U)}{S((R \otimes U) \parr T)} \qquad \mathsf{p}{\downarrow}\,\frac{S\{!(R \parr T)\}}{S(!R \parr ?T)} \qquad \mathsf{w}{\downarrow}\,\frac{S\{\bot\}}{S\{?R\}} \qquad \mathsf{b}{\downarrow}\,\frac{S(?R \parr R)}{S\{?R\}}$$

A derivation Δ is a finite chain of instances of these inference rules. A derivation can consist of just one formula. The top-most formula in a derivation, if present, is called the *premise*, and the bottom-most formula is called its *conclusion*. A derivation Δ whose premise is T, conclusion is R, and inference rules are in $\mathscr{S}$ is written as $\Delta \dfrac{\Big\| \mathscr{S}}{\begin{matrix} T \\ R \end{matrix}}$. A *proof* Π is a finite derivation whose premise is the unit 1.

2.2 Linear Logic Planning

Following [25, 9], a linear logic planning problem $\mathscr{P}$ is given by $\langle \mathcal{I}, \mathcal{G}, \mathscr{A} \rangle$ where $\mathcal{I} : \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$ is a multiset [3] of fluents called the *initial state*. The multiset $\mathcal{G} : \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$ of fluents is the *goal state*. $\mathscr{A}$ is a finite set of *actions* of the form $\mathsf{a} \ : \ \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$, where $\dot{\{}\, c_1, \ldots, c_p \,\dot{\}}$ and $\dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$ are multisets of fluents called *conditions* and *effects*, respectively, and a is the *name* of the action. Action a is applicable to a state $\mathcal{S}$ iff $\dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \,\dot{\subseteq}\, \mathcal{S}$. The application of such an action a to a state $\mathcal{S}$ is defined by the function Φ, where it is applicable, as $\Phi(\mathsf{a}, \mathcal{S}) = (\,\mathcal{S} \,\dot{-}\, \dot{\{}\, c_1, \ldots, c_p \,\dot{\}}\,) \,\dot{\cup}\, \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$.

A goal $\mathcal{G}$ is satisfied iff there is a *plan* P, i.e., a sequence of actions $\mathsf{P} = \langle \mathsf{a}_1; \ldots; \mathsf{a}_k \rangle$ such that $\Phi(\mathsf{a}_k, \ldots, \Phi(\mathsf{a}_1, \mathcal{I}) \ldots) = \mathcal{G}$. Then, we say P *transforms the initial state* $\mathcal{I}$ *into the state* $\mathcal{G}$. If there exists such a plan P then P is a solution for the planning problem $\mathscr{P}$. Then we say P *solves* $\mathscr{P}$. We denote the empty plan with $\circ$. If it is more convenient, $\Phi(\mathsf{a}_k, \ldots, \Phi(\mathsf{a}_1, \mathcal{I}) \ldots)$ is abbreviated with $\Phi(\mathsf{P}, \mathcal{I})$. The *length of a plan* is the number of actions in that plan.

[3] Multisets are denoted by the curly brackets "$\dot{\{}\ \ \dot{\}}$". $\dot{\cup}$, $\dot{-}$ and $\dot{\subseteq}$ denote the multiset operations corresponding to the usual set operations $\cup$, $-$ and $\subseteq$, respectively.

Example 1. Consider the following planning problem: the actions α and β, respectively, buy an apple for fifty cents and buy a banana for fifty cents, i.e., $\mathscr{A} = \{\, \alpha : \dot{\{}\, f \,\dot{\}} \rightarrow \{\, a \,\}\,,\ \beta : \dot{\{}\, f \,\dot{\}} \rightarrow \dot{\{}\, b \,\dot{\}}\,\}$. The initial state and goal state are $\mathcal{I} = \dot{\{}\, f, f \,\dot{\}}$ and $\mathcal{G} = \dot{\{}\, a, b \,\dot{\}}$, respectively. The solutions for this planning problem are the plans $\langle \alpha; \beta \rangle$ and $\langle \beta; \alpha \rangle$. A plan which is executable at the initial state, however not a solution for this problem is the plan $\langle \alpha; \alpha \rangle$.

It is important to observe that the explicit representation of resources, given by the multiset representation, demonstrates that there are no resource conflicts between the actions α and β in the solution plans above. This observation permits the parallel execution of these two actions when there is no hardware constraints. However, in the encodings of linear logic planning, e.g, in [25, 9, 18], plans are extracted by sequentially reading the proper axioms corresponding to actions from the leaves of the proof tree, constructed by using the cut-rule. This does not allow to observe such a parallel execution semantics and breaks the cut-elimination property (see, e.g., [8, 31]).

Let us now present our encoding of the planning problems in multiplicative exponential linear logic, which allows the construction of cut-free proofs.

Definition 1 *Let* $\mathsf{a} = \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$ *be an action. An* action formula for a *(A_{a}) is of the form* $?\,(\bar{c}_1 \otimes \ldots \otimes \bar{c}_p \otimes (e_1 \parr \ldots \parr e_q))$. *A* problem formula *is of the form* $!\,(r_1 \parr \ldots \parr r_m \parr (\bar{t}_1 \otimes \ldots \otimes \bar{t}_n))$.

Definition 2 *Given a planning problem* $\mathscr{P} = \langle\, \mathcal{I}, \mathcal{G}, \mathscr{A} \,\rangle$ *where* $\mathcal{I} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$ *is the initial state,* $\mathcal{G} = \dot{\{}\, t_1, \ldots, t_n \,\dot{\}}$ *is the goal state, and* $\mathscr{A}$ *is a set of actions.*

$$(?A_1 \parr \ldots \parr ?A_s \parr !\,(r_1 \parr \ldots \parr r_m \parr (\bar{t}_1 \otimes \ldots \otimes \bar{t}_n)))$$

is the planning problem formula (ppf) that corresponds to $\mathscr{P}$ *where* $A_1, \ldots, A_s$ *are action formulae for all the actions* $\mathsf{a} \in \mathscr{A}$.

Example 2. When we consider the planning problem of Example 1, we obtain the planning problem formula $?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes a) \parr !\,(f \parr f \parr (\bar{a} \otimes \bar{b}))$.

Lemma 1. *(i.) The rule action below is derivable (sound) in* ELS. *(ii.) Let* $a : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$ *be an action, and* $\mathcal{S} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$ *and* $\mathcal{S}' = \dot{\{}\, t_1, \ldots, t_n \,\dot{\}}$ *be states. For some formulae R, T, and E*

$$\Phi(a, \mathcal{S}) = \mathcal{S}' \quad \textit{iff} \quad \mathsf{action}\ \frac{(\,?(\bar{c}_1 \otimes \ldots \otimes \bar{c}_p \otimes E) \parr !\,(t_1 \parr \ldots \parr t_n \parr R) \parr T\,)}{(\,?(\bar{c}_1 \otimes \ldots \otimes \bar{c}_p \otimes E) \parr !\,(r_1 \parr \ldots \parr r_m \parr R) \parr T\,)} \quad .$$

Lemma 2. *The following rule is derivable (sound) in* ELS.

$$\mathsf{termination}\ \frac{1}{(?A_1 \parr \ldots \parr ?A_s \parr !\,(g_1 \parr \ldots \parr g_m \parr (\bar{g}_1 \otimes \ldots \otimes \bar{g}_m)))}$$

Theorem 3 *Let* $\mathcal{P} = (?A_1 \parr \ldots \parr ?A_s \parr !\,(r_1 \parr \ldots \parr r_m \parr (\bar{t}_1 \otimes \ldots \otimes \bar{t}_n)))$ *be a ppf that corresponds to a planning problem* $\mathscr{P}$*. There is a proof with* k *number of applications of the* $\mathsf{p}\!\downarrow$ *rule iff there is a plan* p *with length* k *that solves the planning problem* $\mathscr{P}$*, where* $\mathcal{I} = \{\, r_1, \ldots, r_m \,\}$ *and* $\mathcal{G} = \{\, t_1, \ldots, t_n \,\}$*.*

Theorem 3 states the equivalence of existence of a plan solving a planning problem with the existence of a proof of the encoding of this planning problem. However, by resorting to this theorem, and Lemma 1 and Lemma 2, we can use the inference rules action and termination as the operational semantics of a planner: these inference rules can be used as machine instructions in an implementation for searching for plans.

Example 3. A proof of the planning problem of Example 1 can be constructed bottom-up as follows. The shaded regions denote the action formula being used at every inference step and the name of the action is displayed on the right-hand side of the inference rules. Then the plan can be extracted by reading these names bottom-up. Thus, the proof below reads the plan $\langle \alpha; \beta \rangle$.

$$\dfrac{\dfrac{\dfrac{1}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(a \parr b \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{termination}}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(a \parr f \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{action}\ \beta}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(f \parr f \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{action}\ \alpha$$

3 Independence and Causality in Plans

Given that a planning problem has a solution, the ppf of this planning problem can be proved in many different ways. Although these different proofs are distinct syntactic objects, they can be considered equivalent, because they share the instances of the rule $\mathsf{ai}\!\downarrow$ applied to the same pairs of atoms. Such an equivalence can be observed, for example, when two multiplicative linear logic proofs are mapped to the same proof net [8] or they can be decomposed to the same proof by permutation of the inference rules [31].

Example 4. We can construct the following proof which is a different syntactic object from the proof in Example 3, however the $\mathsf{ai}\!\downarrow$ rule instances, hidden in the instances of the rules action and termination in the proof below (see Lemma 1 and Lemma 2), are identical in these two proofs.

$$\dfrac{\dfrac{\dfrac{1}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(a \parr b \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{termination}}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(f \parr b \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{action}\ \alpha}{?(\bar{f} \otimes a) \parr ?(\bar{f} \otimes b) \parr !\,(f \parr f \parr (\bar{a} \otimes \bar{b}))}\ \mathsf{action}\ \beta$$

In this section, by using this idea, we present an algorithm for extracting partial order plans, which exhibit a concurrency semantics, from the proofs of the ppf.

Definition 4 *Let Π be a proof* $\rho\,\dfrac{S\{T\}}{S\{R\}}\,{}^{\Pi'\!\!\prod\!\!\mathsf{ELS}}$ *where the atoms in an action formula are labelled with the name of that action. Furthermore, whenever there is an application of the rule* $\mathsf{b}{\downarrow}$*, the labels of the atoms in the premise, which are copied, are extended with a natural number that does not occur with the same action name elsewhere in the proof. Similarly, in a problem formula, all the positive and negative atoms are labelled with* init *and* goal*, respectively. Let Label denote the set of all the labels occurring in Π. The function μ on Π is defined as follows. If $\Pi = 1$, then $\mu(\Pi) = \emptyset$. Otherwise,*

- *if ρ is the application of a rule other than* $\mathsf{ai}{\downarrow}$ *then $\mu(\Pi) = \mu(\Pi')$.*
- *if ρ is the application of the rule* $\mathsf{ai}{\downarrow}$ *where R is the formula $(a_{\mathsf{l}} \parr \bar{a}_{\mathsf{k}})$ for an atom a such that $\mathsf{l}, \mathsf{k} \in Label$, then*

$$\mu(\Pi) = \{\, (\mathsf{l}, \mathsf{k}) \,\} \cup \mu\begin{pmatrix} {}^{\Pi'\!\!\prod\!\!\mathsf{ELS}} \\ S\{1\} \end{pmatrix} .$$

Given a proof Π of $\mathcal{P}$, a constraint set *of Π for $\mathcal{P}$ ($\mathcal{C}_{P,\Pi}$) is given with $\mu(\Pi)$.*

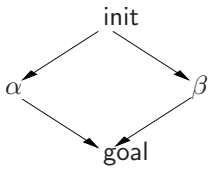
Proposition 1. *For any proof Π of a ppf, $\mu(\Pi)$ terminates in linear time in the number of atoms in Π.*

The constraint sets are obtained by recording the atoms that get annihilated by the $\mathsf{ai}{\downarrow}$ instances. This idea is very similar to using proof nets [8] as a means for identifying classes of equivalent proofs up to permutation of inference rules, or the ideas used to describe classes of proofs that are equivalent upto a geometric criterion similar to proof nets [21]. Because each atom that gets annihilated is produced and consumed by a specific action formula, a constraint set provides an explicit record of causality between the actions producing and consuming each atom in the execution of a plan. Thus, the actions that are partially ordered in a constraint set are independent events in the execution because they do not have any resource conflicts.

Example 5. A proof of Example 1 that is syntactically different from the proofs given in Example 3 and Example 4 is as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{1}{\mathsf{ai}{\downarrow}}}{!\,(b_\beta \parr \bar{b}_{\mathsf{goal}})}}{!\,((a_\alpha \parr \bar{a}_{\mathsf{goal}}) \otimes (b_\beta \parr \bar{b}_{\mathsf{goal}}))}}{!\,((f_{\mathsf{init}} \parr \bar{f}_\beta) \otimes (a_\alpha \parr \bar{a}_{\mathsf{goal}}) \otimes (b_\beta \parr \bar{b}_{\mathsf{goal}}))}}{!\,((f_{\mathsf{init}} \parr \bar{f}_\alpha) \otimes (f_{\mathsf{init}} \parr \bar{f}_\beta) \otimes (a_\alpha \parr \bar{a}_{\mathsf{goal}}) \otimes (b_\beta \parr \bar{b}_{\mathsf{goal}}))}\,\mathsf{s}^{7}}{!\,((\bar{f}_\alpha \otimes a_\alpha) \parr ?(\bar{f}_\beta \otimes b_\beta) \parr !\,(f_{\mathsf{init}} \parr f_{\mathsf{init}} \parr (\bar{a}_{\mathsf{goal}} \otimes \bar{b}_{\mathsf{goal}})))}\,\mathsf{p}{\downarrow}^{2}}{?(\bar{f}_\alpha \otimes a_\alpha) \parr ?(\bar{f}_\beta \otimes b_\beta) \parr !\,(f_{\mathsf{init}} \parr f_{\mathsf{init}} \parr (\bar{a}_{\mathsf{goal}} \otimes \bar{b}_{\mathsf{goal}}))}$$

When we plug into the function μ of Definition 4 this proof, or any of the proofs in Example 3 or Example 4 expanded with respect to Lemma 1 and Lemma 2, we obtain the following partial order.



Proposition 5 *Let $\mathcal{P}$ be a ppf defined on the action set $\mathscr{A}$ and $\mathcal{C}_{\mathcal{P},\Pi}$ be a constraint set of a proof Π for $\mathcal{P}$. (i) $\mathcal{C}_{\mathcal{P},\Pi}$ is antisymmetric. (ii) $\mathcal{C}_{\mathcal{P},\Pi}$ is irreflexive.*

Definition 6 *Let $\mathcal{P}$ be a ppf defined on the action set $\mathscr{A}$ and $\mathcal{C}_{\mathcal{P},\Pi}$ be a constraint set of a proof Π for $\mathcal{P}$. The* concurrent plan of Π for $\mathcal{P}$ *is $\Gamma_{\mathcal{P},\Pi}$ is the transitive reduction (cover relation) of $\mathcal{C}_{\mathcal{P},\Pi}$.*

Definition 7 *A plan P is induced by a strict total order $\prec$ if for any pair $(x, y) \in \prec$, x appears to the left of y in P.*

Theorem 8 *Let $\mathcal{P}$ be a ppf of a planning problem $\mathscr{P}$, Π a proof of $\mathcal{P}$ and $\Gamma_{\mathcal{P},\Pi}$ the concurrent plan of Π for $\mathcal{P}$. For any strict total order $\prec \supseteq \Gamma_{\mathcal{P},\Pi}$, if P is a plan induced by $\prec$ then P solves $\mathscr{P}$.*

This theorem provides an interleaving semantics of the plans computed as proofs of ppf. Let us now give a non-interleaving semantics for these plans.

4 Partial Order Plans with a Concurrency Semantics

In linear logic planning, states are defined over the data structure multiset, actions are considered as multiset rewriting rules. Multiset rewriting is also complete for representing computations of place/transition petri nets [29] (see, e.g., [4, 12]). In such an encoding, the multiset rewrite rules represent the possible firings of the transitions of a petri net. The places of the net are represented by elements of multisets. Such a view allows to consider a planning problem as the reachability problem of the corresponding petri net and vice versa.

4.1 Planning and Concurrency

When planning problems are considered from the point of view of concurrent computations, e.g., as those in petri nets, due to the explicit representation of resources, multiset rewriting planning allows to observe true concurrency in the computations: in a language with true concurrency, when two actions are partially ordered, the outcome of their execution in parallel is same as the outcome of their execution in either order. Because the explicit treatment of resources provides a representation of independence and causality, when two actions are partially ordered, in an execution that involves both of these actions, they are independent in terms of the resources that they require to be executed. Thus,

their parallel composition results in an action that has the same effect as their execution in any order. It becomes possible to define the parallel composition of two actions by taking the multiset-union of their condition and effect multisets [14]. Here, we speak about concurrency, in contrast to only parallelism, because the common predecessors and successors of the two composed actions provide a synchronisation mechanism when they are considered as points in time.

Example 6. In the planning problem of Example 1, we can compose the two actions α and β and obtain the action $\dot{\{}\, f, f \,\dot{\}} \to \dot{\{}\, a, b \,\dot{\}}$, which corresponds to concurrent executions of these two actions. These two actions are then synchronised by their predecessor init and successor goal.

However, when planning problems are modelled in common AI planning formalisms, that are based on properties instead of resources, e.g., STRIPS [5], it is not always possible to observe true concurrency in the partial order plans of these languages, e.g., UCPOP [28] and Graphplan [1]. A simple modification of the famous dining philosophers problem is helpful to see the reason for this.

Example 7. There are two hungry philosophers, a and b, sitting at a dinner table. In order for a philosopher to eat, she must have a fork. However, there is only one fork on the table. The problem consists in finding a plan where both philosophers have eaten. The solution of this problem is a plan in which a and b eat in either order. A plan where a and b eat concurrently cannot be a solution for this problem, because a and b cannot have the fork at the same time. Because the fork is a resource, which cannot be shared, eating of one is dependent on the others finishing eating and leaving the fork. Hence, these two actions can be executed in either order but not in parallel. A simple encoding of this scenario as a planning problem allows to observe such a semantics: let $\mathcal{I} = \dot{\{}\, h_a, h_b, f \,\dot{\}}$, $\mathcal{G} = \dot{\{}\, e_a, e_b, f \,\dot{\}}$ and $\mathscr{A} = \{\ \mathsf{a} : \dot{\{}\, h_a, f \,\dot{\}} \to \dot{\{}\, e_a, f \,\dot{\}}\,, \quad \mathsf{b} : \dot{\{}\, h_b, f \,\dot{\}} \to \dot{\{}\, e_b, f \,\dot{\}}\ \}$. In the encoding above, for a philosopher x, the resource h_x denotes that x is hungry and e_x denotes that x has eaten. f denotes the resource fork. The actions a and b can be executed in either order. However, their parallel composition results in the action $[\mathsf{a}, \mathsf{b}] : \dot{\{}\, h_a, h_b, f, f \,\dot{\}} \to \dot{\{}\, e_a, e_b, f, f \,\dot{\}}$ which requires two instances of the resource f in order to be executed. Thus the parallel composition of these two actions cannot be executed in the initial state $\mathcal{I}$. An encoding of this problem by means of properties, in a propositional language, in a way which delivers such a semantics is not straight-forward, if not impossible.

Due to the explicit treatment of resources, in contrast to the partial order planners in the literature, such as, UCPOP or Graphplan, the approach of the present paper respects the dependency and causality between actions in a planning domain, and results in a non-interleaving, behavioural concurrency semantics, namely, labelled event structure semantics.

4.2 Labelled Event Structure Semantics of Planning Problems

Labelled event structures (LES) is a non-interleaving branching-time behavioural model of concurrency [34]. An interleaving model of concurrency is equipped

with an expansion law that identifies parallel composition by means of choice and sequential composition. In an interleaving model, parallel composition of two events indicates that these events can take place in either order. A model for concurrency without such an expansion law is said to be a non-interleaving model: when two events are composed in parallel they can take place simultaneously or in either order. In such a view of the systems, the independence and causality between the events of the system is central. In a LES the causality between actions is captured in terms of their dependencies in a partial order.

Apart from the causality given by a partial order, in a LES, the nondeterminism in the computation is captured by a conflict relation, which is a symmetric irreflexive relation of events. In a planning perspective, this corresponds to actions that are applicable in the same state, but are in conflict. When two actions are in conflict with each other, execution of one of them instead of the other determines a different state space ahead. This provides a branching-time model of the possible computations.

By using the operational semantics given by the rule action, in [13], we provide a procedure for obtaining a LES from the specification of a planning problem: for each planning problem, we obtain a labelled transition system determined by the possible applications of the rule action. We then apply the standard techniques for obtaining a LES from these transition systems (see, e.g., [34, 10]). The LES obtained takes all the possible computations in the planning domain into consideration and reflects the plans that are in conflict with each other. Because of the explicit treatment of resources in linear logic, the computations in the LES reflect the independence and causality between actions of the computation. The concurrent plans of Definition 6 reflect this semantics.

Example 8. Consider the planning problem of Example 1. The computations of this system are given as the LES depicted below, where # denotes the conflict relation. The four events are partially ordered because they are causally independent. The events α_1 and β_2 are in conflict because they cannot co-occur. Similarly, the events α_2 and β_1 are in conflict. The events that are not in conflict and causally independent can co-occur, e.g., α_1 and α_2 or α_1 and β_1.

Example 9. There are two tables. On Table 1 there are four blocks which are stacked on top of each other as shown on the left-hand side of the Figure 1. The only available action takes a block from Table 1 and puts it on Table 2. The goal of the problem is moving three of the blocks from Table 1 to Table 2. Because block a is stacked on blocks c and d, blocks c and d cannot be moved before block a. Similarly, block d cannot be moved before block b.

The LES displaying the causality and independence of the planning problem is on the right-hand side of Figure 1. Each event there represents moving the corresponding block. Events a and b are independent, however b and c are also independent although event c causally requires event a in order to occur. The events c and d are independent but they cannot co-occur in an execution of the system because they are in conflict. The possible concurrent plans are the two maximal

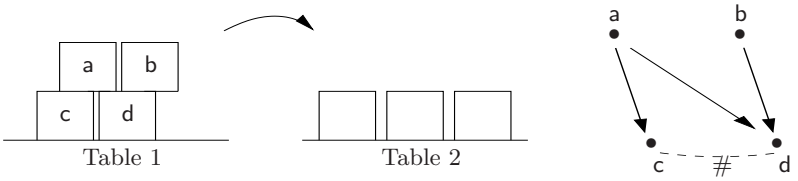


Fig. 1. A planning problem and the corresponding LES

conflict-free partial order in LES, i.e., $\{ (\mathsf{init}, \mathsf{a}), (\mathsf{init}, \mathsf{b}), (\mathsf{b}, \mathsf{goal}), (\mathsf{a}, \mathsf{c}), (\mathsf{c}, \mathsf{goal}) \}$ and $\{ (\mathsf{init}, \mathsf{a}), (\mathsf{init}, \mathsf{b}), (\mathsf{a}, \mathsf{d}), (\mathsf{b}, \mathsf{d}), (\mathsf{d}, \mathsf{goal}) \}$. The interleavings of these concurrent plans are given by the plans $\langle \mathsf{a}; \mathsf{c}; \mathsf{b} \rangle$, $\langle \mathsf{b}; \mathsf{a}; \mathsf{c} \rangle$, $\langle \mathsf{a}; \mathsf{b}; \mathsf{c} \rangle$, $\langle \mathsf{a}; \mathsf{b}; \mathsf{d} \rangle$ and $\langle \mathsf{b}; \mathsf{a}; \mathsf{d} \rangle$, which solve this problem.

5 Relation to Other Work

The reachability problem in petri nets is known to be EXPSPACE-hard [22]. Thus, the encoding of petri nets as multiset rewriting systems in multiplicative exponential linear logic delivers the lower bound of this logic to be EXPSPACE-hard [23]. When the complexity of a language is seen as a measure of expressive power, this also sets the scene for the expressive power of the propositional languages based on multiset rewriting in comparison to propositional languages based on STIRPS: given that planning in STRIPS is PSPACE-complete [3], because PSPACE is a strict subset of EXPSPACE multiset rewriting is strictly more expressive than propositional languages based on STRIPS. In order to achieve the same expressive power, these languages must be enriched with a constant-only first order language, i.e., DATALOGSTRIPS. However, a characterisation of STRIPS in multiset planning is possible (see, e.g., [20]).

If we consider the planning languages based on properties, e.g., ADL [27], we see that any planning problem expressed in these languages can be expressed as a multiset planning problem: [32] shows that multiset planning languages can be employed to encode the domain descriptions of the action description language $\mathcal{A}$ [6]. As it is stated in [7], because the action description language $\mathcal{A}$ is equivalent to the propositional fragment of the planning language ADL the result of [32] also implies that the multiset rewriting approach can be used for ADL domains.

For a more extensive survey on reasoning about actions, planning and concurrency we refer to [13].

6 Discussion

In [14, 13], we have introduced a deductive language [4] for multiset planning within an extension of multiplicative exponential linear logic with a noncommutative self-dual operator [11]. In this language, the sequential composition

[4] Prototype implementations of planners based on this approach, mainly in Maude language, are available at `http://www.doc.ic.ac.uk/~ozank/maude_cos.html`

of the actions is represented by means of the non-commutative self-dual logical operator, whereas the parallel composition of the actions is naturally mapped to the commutative par operator of linear logic. Thus, by means of this language parallel and sequential composition of actions and plans can be represented at the same logical level as in process algebra and logical reasoning can be performed on these plan expressions. Ongoing work includes using the ideas of this paper to provide an event structure semantics to this deductive language with a proof theoretical operational semantics. This language should then benefit from a rich arsenal of tools and techniques that are imported from the both fields of planning and concurrency, and find applications, e.g., in modelling biological systems as complex reactive systems [15].

Acknowledgements: The author would like to thank Alessio Guglielmi, Steffen Hölldobler, Luca Cardelli, Max Kanovich and anonymous referees for valuable comments and improvements.

References

1. A. Blum and M. Furst. Fast planning through planning graph analysis". *Artificial Intelligence*, 90:281–300, 1997.
2. C. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.
3. T. Bylander. Complexity results for serial decomposability. In *Proc. of the Tenth National Conf. on AI (AAAI-92)*, pages 729–734, San Jose, 1992. AAAI Press.
4. Iliano Cervesato. Petri nets and linear logic: a case study for logic programming. In *Proceedings of the Joint Conference on Declarative Programming: GULP-PRODE'95*, Marina di Vietri, Ital, 1995.
5. R. E. Fikes and H. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–205, 1971.
6. Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17(2/3–4):301–321, 1993.
7. Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2 (3–4):193–210, 1998.
8. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
9. G. Große, S. Hölldobler, and J. Schneeberger. Linear deductive planning. *Journal of Logic and Computation*, 6 (2):233–262, 1996.
10. Alessio Guglielmi. *Abstract Logic Programming in Linear Logic Independence and Causality in a First Order Calculus*. PhD thesis, Universita di Pisa, 1996.
11. Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007.
12. K. Ishihara and K. Hiraishi. The completeness of linear logic for petri net models. *Logic Journal of IGPL*, 9(4):549–567, 2001.
13. Ozan Kahramanoğulları. *Nondeterminism and Language Design in Deep Inference*. PhD thesis, Technische Universität Dresden, 2006.
14. Ozan Kahramanoğulları. Towards planning as concurrency. In *Proceedings of the IASTED International Conference on Artificial Intellgence and Applications, AIA 2005*, pages 387–393, Innsbruck, Austria, 2005.
15. Ozan Kahramanoğulları. A deductive compositional approach to petri nets for systems biology. Poster presentation at the CMSB Conference, submitted, 2007.

16. Ozan Kahramanoğulları. System BV is NP-complete. *Annals of Pure and Applied Logic*, 152(1–3):107–121, 2008.
17. M. I. Kanovich and J. Vauzeilles. The classical AI planning problems in the mirror of horn linear logic: semantics, expressibility, complexity. *Mathematical Structures in Computer Science*, 11(6):689–716, 2001.
18. M. I. Kanovich and J. Vauzeilles. Strong planning under uncertainty in domains with numerous but identical elements (a generic approach). *Theoretical Computer Science*, 379:84–119, 2007.
19. Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In *Artificial Intelligence Planning Systems*, pages 98–103, 1994.
20. Peep Küngas. Linear logic for domain-independent AI planning (extended abstract). In *Proc. of Doctoral Consortium at 13th Int. Conf. on Automated Planning and Scheduling, ICAPS 2003*, pages 68–72, Trento, Italy, 2003.
21. François Lamarche and Lutz Straßburger. Naming proofs in classical propositional logic. In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications, TLCA 2005*, volume 3461 of *LNCS*, pages 246–261. Springer-Verlag, 2005.
22. R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.
23. N. Martí-Oliet and J. Meseguer. From petri nets to linear logic. *Mathematical Structures in Computer Science*, 1:66–101, 1991.
24. N. Mart'i-Oliet and J. Meseguer. Action and change in rewriting logic. In R. Pareschi and B. Fronhofer, editors, *Dynamic Worlds: From the Frame Problem to Knowledge Management*, volume 11–2, pages 1–53. Kluwer Academic, 1999.
25. M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I–II. In *Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 63–75. Springer-Verlag, 1990.
26. Dale Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the 1992 Workshop on Extensions to Logic Programming*, number 660 in LNCS, pages 242–265. Springer-Verlag, 1992.
27. E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachmann, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the First Int. Conf. (KR-89)*, pages 324–332, Toronto, ON, 1989. Morgan Kaufmann.
28. J. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *KR 92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114, 1992.
29. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
30. R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–13. Cambridge, Morgan Kaufmann, 1996.
31. Lutz Straßburger. MELL in the calculus of structures. *Theoretical Computer Science*, 309:213–285, 2003.
32. Michael Thielscher. Representing Actions in Equational Logic Programming. In P. Van Hentenryck, editor, *Proc. of the Int. Conf. on Logic Programming (ICLP)*, pages 207–224, Santa Margherita Ligure, Italy, 1994. MIT Press.
33. Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2):4:1–24, 2006.
34. Glynn Winskel and Morgens Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, volume 4, pages 1–148. Oxford University Press, 1995.