CSTRIPS: Toward Explicit Concurrent Planning

Marcelo Oglietti[1] and Amedeo Cesta[2]

[1] Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica
Via Salaria 113, 00198 Roma, Italy. oglietti@dis.uniroma1.it
[2] ISTC-CNR, National Research Council of Italy
Viale Marx 15, I-00137 Roma, Italy. a.cesta@istc.cnr.it

Abstract. This paper presents a formalism that extends the STRIPS planning language to explicitly deal with concurrent actions. According to this approach, possibly concurrent activities of a domain can be modeled by representing them explicitly as different concurrent threads. The first part of the paper introduces this empowered STRIPS, called Concurrent STRIPS (or CSTRIPS), by giving its syntax and semantics. The second part addresses aspects that help understand the effectiveness of CSTRIPS. It first illustrates a complete example of a quite complex domain in which causal and process reasoning are intertwined, and CSTRIPS is shown to support very well the modeling phase. Then two aspects are addressed which are connected to the computational complexity of reasoning in CSTRIPS: (a) the introduction of the explicit representation of concurrency is shown not to worsen the computational complexity of the problem solving phase; (b) it is shown that in highly parallel domains it is possible to speed up the reasoning by representing the domain as concurrent threads.

1 Introduction

Modeling the intrinsic concurrency of many real world domains is a challenging goal that has been only partially explored in planning. A comment from [5] well explains the lack of research exploration in this direction: "despite the interest in multi-agent applications (...) and the large body of work on distributed multi-agent planning, very little research addresses this basic problem of planning in the context of concurrent interacting actions."

Passive concurrency. On-going actions can be modeled by using a "continuous tense predicate" (e.g. Flying(A)) that mark the action is on-going. When this "continuous tense predicate" holds we have that the action is on-going and otherwise when it is false. Clearly, we need also at least two operators: one that makes the "continuous tense predicate" true for starting the on-going action and another that makes it false to terminate the on-going action. When we have that more than one of these "continuous tense predicates" predicate holds, we simply have more than one on-going action occurring. Hence, this is an extremely simple way of representing the concurrency of actions that has the great advantage of not requiring any change to the planning framework. This approach has been followed inside different planning frameworks, see [25] for the Situation Calculus

(replacing in previous sentences 'predicate' with 'fluent') and [2, 1] for PDDL. The major drawback of this approach is that the many "continuous tense predicates" that represent the concurrency of on-going actions are *passive* and cannot really act by themselves, hence, they are not actions in the usual AI planning sense, e.g. as the instantiated operator in STRIPS. It is only the standard sequence of actions that is active and change the state of the world. Hence, the concurrency is usually *interleaved* in this approach and it is impossible to start two actions simultaneously. We refer to this form of representation of the concurrency as *passive concurrency*. Also important is that, as is noted in [1], this approach is not modular.

ConGolog [9], is a concurrent logic programming language based on the Situation Calculus that represent the concurrency explicitly. Sadly, these programs cannot be synthesized (planned for) because they are macros and do not belong to the first order logic. When dealing with planning, ConGolog programs can be used at most to guide the planner into produce certain plans that respect the constrains imposed by the programs. It is not possible to plan for plans that represent the concurrency of actions as ConGolog programs. The plans obtained are only those normally represented in the Situation Calculus, and hence, the observations made in the previous paragraph hold for the ConGolog approach.

Implicit representation of concurrency. Most of the planning frameworks deal with concurrency *implicitly* as a side effect of the planning phase. Quoting Bäckström, several approaches follow the intuitive idea that "a non-linear plan is a *parallel plan* if any unordered actions can be executed in parallel without interfering with each other." [3]. Among the planning frameworks of this type, some have been well formalized, e.g., ADL [24], HTN [10], PDDL [21, 11] and the SAS family [3]. Even if these frameworks explore very different strategies for planning, all of them represent concurrency implicitly, like in partial order planning. The same is valid for several works oriented to produce practical architectures (see [14, 18] to name a few). Also COLLAGE [16] is an interesting non-traditional *action-based* planner which follows a similar approach. It is a CSP-based planner that divides the problem in regions and uses local search techniques to exploit the structure given by the regions to improve the search costs. Its plans consist of partially ordered actions, i.e. it does not represent concurrency explicitly. This implicit approach has various limitations such as the difficulty to express quantitative synchronization constraints between different actions. For a more complete discussion of these limitations see for example: [6, 7, 12, 17].

Explicit representation of concurrency. A number of sparse approaches go beyond the previous assumptions and propose some kind of explicit representation of concurrency. An example is BTPL [6, 7], an approach that includes an explicit representation of time, and –at some level– of the concurrency of the planning domain, based on the concept of events. Another approach that represent the concurrency explicitly is [5]. Our work is similar to this approach, and due to this we address this subject with more detail in the conclusions.

A different inspiration derives from research on CSP scheduling, e.g. [8, 4], in which the world is modeled as a set of concurrent resources that should accommodate a set of activities satisfying heterogeneous constraints of various complexity. In a modeling perspective, resources are seen here as *concurrent threads*

that evolve in parallel on a timeline and may have synchronization points when particular activities are using them (for example when an activity requires two heterogeneous resources to be executed). *We are interested in studying the use of concurrent threads for planning actions, similarly to the use of resources in CSP scheduling.* HSTS [22], the planner used in the Remote Agent Experiment [23], extends concepts from scheduling to represent planning causal knowledge. In that planner the concurrent threads are called *state variables* and are state transition systems that evolve in parallel and represent complex behavior of physical systems. Synchronization points are then introduced to model *causal constraints* among different domain components. Unfortunately the formal semantics of HSTS is not clearly described, for recent attempts see [13]. Even if similar modeling ideas are used in some formal approaches to verification [1] a formalization of the complete system has not been presented so far.

Our goal. We are interested in developing a formal framework for planning with concurrent threads that we generally refer to as *Explicit Concurrent Planning*. Instead of developing the idea in a non classical planning framework, we are pursuing the idea of integrating concurrent threads within STRIPS. We would like to provide the knowledge engineer with the capabilities to model a domain by explicitly associating different actions with different threads of evolution. The paper introduces a new action theory, its syntax and semantics, as a general theoretical framework for planning in which concurrency is represented explicitly. This theory can be exploited by different types of planners, as pointed out in the sequel of the paper, in particular in the section on future work. In addition, our proposal paves the way to provide theoretical foundations to some of the above mentioned recent explorations of planning with concurrent threads.

The main contribution of this paper is to formally provide a simple and basic extension of STRIPS called **Concurrent STRIPS** (or **CSTRIPS**) which allows us to describe planning domains as collections of concurrent threads. CSTRIPS is an *Explicit Concurrent Planning* framework which explicitly models the concurrent characteristics of a domain as *concurrent threads* in simultaneous parallel evolution. Each concurrent thread is modeled as a set of STRIPS-like operators exclusively associated with it. CSTRIPS is more expressive than STRIPS in the sense that it allows us to explicitly express the concurrent properties of a domain. But how is this improvement reflected in the computational complexity? In this respect the paper answers the following two important questions: (a) does CSTRIPS perform more poorly than STRIPS? The paper shows that it is possible to encode any CSTRIPS planning problems in STRIPS and that in the worst case its reasoning has the same complexity as STRIPS on the expanded instances [2]; (b) are there cases in which the explicit representation of concurrency yields a computational advantage? The paper shows that in certain domains, those in which there are permanently many parallel actions in execution, it is possible to speed up the reasoning by representing the domain as concurrent threads.

[1] Some approaches are closely related with this kind of modeling and used in the analysis of concurrent programs as Concurrent Transition Systems (e.g., see [15]).

[2] This contrasts with what happens in HTN Planning. HTN is another extension of STRIPS also intended to give more structure when specifying planning domains, but –as shown in [10]– it has a greater computational complexity than STRIPS.

The rest of the paper is organized as follows. The second section formally describes the syntax and semantics of CSTRIPS. The third section shows how CSTRIPS captures in a natural way the different features of a complex scenario, the Shared Oil Tank domain. The fourth section explores the complexity issues of CSTRIPS, the relation with STRIPS, and shows how any planner for STRIPS can be used for planning in CSTRIPS. The fifth section shows a basic CSTRIPS property concerning the gain in branching factor in highly parallel domains. A concluding section summarizes the paper, draws some conclusions and highlights some directions for future work.

2 CSTRIPS: Concurrent STRIPS

This section formally defines CSTRIPS (for Concurrent STRIPS). Similarly to STRIPS, we are creating a family of CSTRIPS frameworks. It is possible to obtain different members of the family by specifying restrictions in the expressivity of the language used to represent the preconditions, effects, and the goal, and by the cardinality of the sets of constants, predicates and functions symbols, etc.

The state space. To formalize CSTRIPS we use here a first order language $\mathcal{L}$ defined by using a tuple $(\mathcal{C}, \mathcal{V}, \mathcal{P}_s, \mathcal{F})$ of mutually disjoint sets of constant, variable, predicate and function symbols, to describe the state of the world. In CSTRIPS the State Space is defined, exactly like in STRIPS, by any subset of ground atoms of $\mathcal{L}$, meaning, the subset of ground atoms that hold in that particular state. Hence, the **State Space** is defined by $\mathcal{S} \equiv 2^{\mathcal{P}}$, where $\mathcal{P}$ is the set of all ground atomic predicates in $\mathcal{L}$.

The syntax. Operators are also defined like in STRIPS. We use a different set of symbols $\mathcal{N}$, disjoint with the previous sets, to represent the names of the operators.

Definition 1 *The **Operator Specification Language** is defined by:*

$$\mathcal{L}_{\mathcal{O}} \equiv \{ \alpha \mid \alpha = (\mathrm{n}_\alpha(\overrightarrow{x}) : P(\overrightarrow{x}) \Rightarrow E(\overrightarrow{x})) \}$$

where $\mathrm{n}_\alpha \in \mathcal{N}$ is the operator's name, $\overrightarrow{x}$ are it's parameters (a possibly empty finite sequence of variable symbols), and $P(\overrightarrow{x})$ and $E(\overrightarrow{x})$ are respectively its preconditions and effects: conjunctions of literals whose variables only in $\overrightarrow{x}$.

The main idea underlying CSTRIPS is that there are a *finite number* of **concurrent threads** that evolve simultaneously and in parallel. CSTRIPS *explicitly represents* this fact and assumes that it is necessary to explicitly plan for all these simultaneous threads. A thread is simply characterized by a non empty set of operators that can be applied in that thread. In general not all operators can be applied in any thread, and hence, to specify a domain, we need to specify the non empty set of operators of each thread. The **CSTRIPS Concurrent Thread Specification Language**, denoted $\mathcal{L}_{\mathcal{T}}$, is defined by: $\mathcal{L}_{\mathcal{T}} \equiv 2^{\mathcal{L}_{\mathcal{O}}} \setminus \emptyset$. The language to specify domains composed by n threads is defined as the Cartesian product of any n threads definitions. Hence, a language to specify domains with a number of threads is simply the union over any possible n.

Definition 2 *The **CSTRIPS Domain Specification Language** is:*

$$\mathcal{L}_{\mathcal{D}_{CSTRIPS}} \equiv \bigcup_{\forall n \in \mathbb{N}} \mathcal{L}_{\mathcal{D}_{\mathrm{CSTRIPS(n)}}} \text{, with : } \mathcal{L}_{\mathcal{D}_{\mathrm{CSTRIPS(n)}}} \equiv \bigcup_{\forall \mathcal{T}_1, \ldots, \mathcal{T}_n \in \mathcal{L}_{\mathcal{T}}} \{ \mathcal{T}_1 \times \ldots \times \mathcal{T}_n \}$$

For simplicity, when the context avoids any confusion, we write $\mathcal{L}_{\mathcal{D}}$ in the place of $\mathcal{L}_{\mathcal{D}_{CSTRIPS}}$ and $\mathcal{L}_{\mathcal{D}(n)}$ in the place of $\mathcal{L}_{\mathcal{D}_{\text{CSTRIPS(n)}}}$.

Hence, in CSTRIPS, a **domain** is defined as any element of this language: $\mathcal{D} \in \mathcal{L}_{\mathcal{D}}$. $(\mathcal{D})_i$ denotes the *ith* set of the Cartesian product that defines $\mathcal{D}$, called the ***ith* thread** of $\mathcal{D}$. Given $\mathcal{D} \in \mathcal{L}_{\mathcal{D}}$ we say that any of its elements $\overrightarrow{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathcal{D}$ is an ***n*-threads operator**. An n-threads operator $\overrightarrow{\alpha}$ represents n operators that are simultaneously applied in parallel, each of its components corresponds to the definition of a STRIPS operator. A domain has all possible n-threads operators that can be made by combining n operators each one taken from a different thread. The *ith* component is necessarily an operator of the *ith* thread of $\mathcal{D}$ and represents what $\overrightarrow{\alpha}$ does in the *ith* thread. We denote $\overrightarrow{\alpha}\,\overrightarrow{\theta}$ an **instantiated *n*-threads operator**, meaning the result of the application to every component of $\overrightarrow{\alpha}$ of the corresponding substitution, replacing in each $(\overrightarrow{\alpha})_i$ all its parameters by the corresponding ground terms given by $(\overrightarrow{\theta})_i$, e.g.: $\overrightarrow{\alpha}\,\overrightarrow{\theta} = (\alpha_1\theta_1, \ldots, \alpha_n\theta_n)$.

For any operator α we respectively denote with $\text{Pre}(\alpha), \text{Pre}^+(\alpha)$ and $\text{Pre}^-(\alpha)$ all/positive/negative literals that appear in its preconditions. For any concurrent thread $\mathcal{T} \in \mathcal{L}_{\mathcal{T}}$ we denote by $\text{Pre}(\mathcal{T}), \text{Pre}^+(\mathcal{T})$ and $\text{Pre}^-(\mathcal{T})$ the sets that contain respectively all/positive/negative literals in the preconditions of all the operators in that thread. Similarly, for any n-threads operator $\overrightarrow{\alpha}$ we denote by $\text{Pre}(\overrightarrow{\alpha}), \text{Pre}^+(\overrightarrow{\alpha})$ and $\text{Pre}^-(\overrightarrow{\alpha})$ the corresponding sets but collecting the literals of the components of $\overrightarrow{\alpha}$. Equivalently with respect to the effects we denote: $\text{Eff}(\alpha)$, $\text{Eff}^+(\alpha)$, $\text{Eff}^-(\alpha)$, $\text{Eff}^+(\mathcal{T})$, $\text{Eff}^-(\mathcal{T})$, $\text{Eff}(\mathcal{T})$, $\text{Eff}^+(\overrightarrow{\alpha})$, $\text{Eff}^-(\overrightarrow{\alpha})$ and $\text{Eff}(\overrightarrow{\alpha})$.

Definition 3 *$I_{\text{CSTRIPS}} \equiv \mathcal{L}_{\mathcal{D}} \times \mathcal{S} \times \mathcal{L}_{\mathcal{G}}$ defines the* ***set of all the CSTRIPS Planning Problem instances****. $\mathcal{L}_{\mathcal{G}}$ is the* ***goal specification language****.*

Hence, $x \in I_{\text{CSTRIPS}}$ means that x is any instance of the CSTRIPS Planning Problem. We do not commit CSTRIPS to a particular goal specification language because this is not essential for the present paper. $\mathcal{L}_{\mathcal{G}}$ can be as simple as a conjunctive list of literals or a more complex language, but it is restricted by the semantics we give below to express only **reachability goals**, not extended goals. We denote with $\mathcal{D}(x)$ the planning domain specification of x, respectively with $s_0(x)$ and $\mathcal{G}(x)$ its initial state and goal.

The semantics. The semantics of CSTRIPS is developed with the idea of eliminating ill defined n-threads operators as much as possible. For example, in STRIPS an operator does not make sense if its effect is to add and delete the same atomic proposition; hence this case is explicitly excluded when defining its semantics. In CSTRIPS, due to the fact that we apply n-threads simultaneously in parallel, we need to exclude more complex problematic situations.

Definition 4 *Given any domain $\mathcal{D} \in \mathcal{L}_{\mathcal{D}}$ of n concurrent threads and any of its instantiated n-threads operators $\overrightarrow{\alpha}\,\overrightarrow{\theta} = (\alpha_1\theta_1, \ldots, \alpha_n\theta_n) \in \mathcal{E}(\mathcal{D})$, we say that $\overrightarrow{\alpha}\,\overrightarrow{\theta}$ is* ***consistent*** *iff:*

1.No competing preconditions: $\quad \text{Pre}^+(\overrightarrow{\alpha}\,\overrightarrow{\theta}) \cap \text{Pre}^-(\overrightarrow{\alpha}\,\overrightarrow{\theta}) = \emptyset$

2.No inconsistent effects: $\quad \text{Eff}^+(\overrightarrow{\alpha}\,\overrightarrow{\theta}) \cap \text{Eff}^-(\overrightarrow{\alpha}\,\overrightarrow{\theta}) = \emptyset$

Definition 5 *For any domain of n concurrent threads:*

- *Any consistent instantiated n-threads operator* $\overrightarrow{\alpha}\,\overrightarrow{\theta}$ *of this domain is* ***executable*** *in a state* $s \in \mathcal{S}$*, iff:*
$$(\ \forall a \in \mathrm{Pre}^{+}(\overrightarrow{\alpha}\,\overrightarrow{\theta})\ .\ a \in s\ )\ \ \wedge\ \ (\ \forall a \in \mathrm{atoms}(\mathrm{Pre}^{-}(\overrightarrow{\alpha}\,\overrightarrow{\theta}))\ .\ a \notin s\ )$$
- ***Applying*** *any executable operator* $\overrightarrow{\alpha}\,\overrightarrow{\theta}$ *in state* s ***results*** *in state* s'*, denoted* $s \models^{\overrightarrow{\alpha}\,\overrightarrow{\theta}} s'$*, iff we have:*
$$s' \ = \ s \cup \mathrm{Eff}^{+}(\overrightarrow{\alpha}\,\overrightarrow{\theta}) \setminus \mathrm{atoms}(\mathrm{Eff}^{-}(\overrightarrow{\alpha}\,\overrightarrow{\theta}))$$

For any domain $\mathcal{D} \in \mathcal{L}_{\mathcal{D}}$ of n concurrent threads, an n**-threads plan** $\mathbb{P}$ is defined as any finite sequence $(\overrightarrow{\alpha}_1\,\overrightarrow{\theta}_1, \ldots, \overrightarrow{\alpha}_m\,\overrightarrow{\theta}_m)$ of instantiated consistent n-threads operators of $\mathcal{D}$. plans($\mathcal{D}$) denotes the **set of all** n**-threads plans** in $\mathcal{D}$. Given any $\mathbb{P} = (\overrightarrow{\alpha}_1\,\overrightarrow{\theta}_1, \ldots, \overrightarrow{\alpha}_m\,\overrightarrow{\theta}_m) \in \mathcal{D}$, we say that its **length**, denoted $\|\mathbb{P}\|$, is the length of the sequence of instantiated n-threads operators: $\|\mathbb{P}\| = m$. Notice that a plan $\mathbb{P}$ is an $n \times m$ matrix, where n is the the number of concurrent threads of the domain and m is its length.

Definition 6 *Given* $\mathcal{D} \in \mathcal{L}_{\mathcal{D}}$ *a domain of n concurrent threads we say that* $\mathbb{P} = (\overrightarrow{\alpha}_1\,\overrightarrow{\theta}_1, \ldots, \overrightarrow{\alpha}_m\,\overrightarrow{\theta}_m) \in \mathrm{plans}(\mathcal{D})$ *is a plan* ***executable*** *in state* $s_0 \in \mathcal{S}$ *iff:*

$$s_0 \ \models^{\overrightarrow{\alpha}_1\,\overrightarrow{\theta}_1}\ s_1 \ \models^{\overrightarrow{\alpha}_2\,\overrightarrow{\theta}_2}\ s_2 \ \ldots \ \models^{\overrightarrow{\alpha}_m\,\overrightarrow{\theta}_m}\ s_m$$

and hence each $\overrightarrow{\alpha}_i\,\overrightarrow{\theta}_i$ *is executable in state* $s_{i-1} \in \mathcal{S}$ *for* $i = 1 \ldots m$*. In this case we say that* ***applying*** $\mathbb{P}$ *in state* s_0 ***results*** *in state* s_m*, denoted:* $s_0 \models^{\mathbb{P}} s_m$

To determine when an n-threads plan is a solution, we need to provide a semantics for the goal specification language $\mathcal{L}_{\mathcal{G}}$. We are only dealing with reachability goals, hence we know that the goal exclusively depends on the singular state at which we are evaluating it. Therefore, we only need to specify the function: $\eta : \mathcal{L}_{\mathcal{G}} \times \mathcal{S} \to \{\top, \bot\}$ that defines for each pair $(\mathcal{G}, s)$ of a reachability goal and a state, when $\mathcal{G}$ holds in s, i.e. when $\eta(\mathcal{G}, s) = \top$. As an example, we give the η for the case when $\mathcal{L}_{\mathcal{G}}$ is any conjunction of literals of $\mathcal{L}$, i.e. $\mathcal{L}_{\mathcal{G}} = \mathrm{literals}(\mathcal{L})$.

$$\eta(\mathcal{G}, s) \ \equiv \ \begin{cases} \top & \text{if } \ (\mathrm{literals}^{+}(\mathcal{G}) \subseteq s_m) \ \wedge \\ & \quad\ (\mathrm{atoms}(\mathrm{literals}^{-}(\mathcal{G})) \cap s_m \ = \ \emptyset) \\ \bot & \text{o.w.} \end{cases}$$

Definition 7 *Given any instance* $x \in I_{\mathrm{CSTRIPS}}$ *of the CSTRIPS planning problem of n concurrent threads an n-threads plan* $\mathbb{P} = (\overrightarrow{\alpha}_1\,\overrightarrow{\theta}_1, \ldots, \overrightarrow{\alpha}_m\,\overrightarrow{\theta}_m) \in \mathrm{plans}(\mathcal{D}(x))$ *is* ***a solution*** *of instance x iff :*
$$s_0(x) \ \models^{\mathbb{P}}\ s_m \quad \wedge \quad \eta(\mathcal{G}(x), s_m) = \top$$

3 A Complete Example

The domain partially depicted in Figure 1 shows agents that can pursue different tasks in parallel. Any agent has the capability to execute two different sets of actions: (a) they can operate some valves controlling the input/output flow of oil exchanged between the agents and a shared oil tank depicted in the figure;

(b) they can execute "personal" tasks that we do not explicitly represent here except for the fact that they may have as a precondition a certain precise range of oil flow exchanged with the central tank, e.g., the agents may need a certain level of oil consumption to execute certain activities. The goal is to accomplish something with the personal tasks and simultaneously keep the level of the tank within a certain range.

This domain can be modeled in CSTRIPS by using two concurrent threads for each: one thread has only the operators that control the flow of exchange with the tank, meanwhile the other thread models all the other actions that the agent can do in parallel with the control of the flow. Additionally, a concurrent thread models the evolution of the oil level in the tank.

According to Figure 1 our domain consists of three agents (named $A1, A2$, and $A3$) and a tank with four valves: I, the only input valve; $O1, O2, andO3$, the three output valves. Dotted arrows show that "agent y may influence domain property x". For example, depending on the tasks they are executing the three agents consume oil so they access $O1, O2$,and $O3$ correspondingly. Only one of them,$A1$, produces oil and, as a consequence, is able to regulate the level of production with the input valve I.

In order to highlight the main characteristics of an explicit representation of concurrency we intentionally have modeled the example with a very restricted formulation of CSTRIPS which does not allow us to use either functions or numbers. The set of constants is:

$$\mathcal{C} = \{0, 1, \ldots, 9, \mathrm{L00}, \mathrm{L01}, \ldots, \mathrm{L99}, \mathrm{L100}\}$$

The first constants from 0 to 9 will be used to represent the percentage of input/output of any valve. The other constants, from L00 to L100 are used to represent the value of the level of oil in the tank.

Our domain specification has the following seven concurrent threads, two for each of the agents $A1, A2$ and $A3$ plus one for the tank level:

$$\mathcal{D} \equiv \mathcal{T}_1 \times \mathcal{T}_2 \times \mathcal{T}_3 \times \mathcal{T}_4 \times \mathcal{T}_5 \times \mathcal{T}_6 \times \mathcal{T}_7$$

where:

$\mathcal{T}_1 = \{\alpha_{1,1}, ..., \alpha_{1,m_1}\}, \quad \mathcal{T}_2 = \{\alpha_{2,1}, ..., \alpha_{2,m_2}\}, \quad \mathcal{T}_3 = \{\alpha_{3,1}, ..., \alpha_{3,m_3}\}$
$\mathcal{T}_4 = \{(\mathrm{ctrlI}(x) : \emptyset \Rightarrow \{\mathrm{I}(x)\}), (\mathrm{ctrlO1}(x) : \emptyset \Rightarrow \{\mathrm{O1}(x)\}), (w : \emptyset \Rightarrow \emptyset)\}$
$\mathcal{T}_5 = \{(\mathrm{ctrlO2}(x) : \emptyset \Rightarrow \{\mathrm{O2}(x)\}), (w : \emptyset \Rightarrow \emptyset)\}$
$\mathcal{T}_6 = \{(\mathrm{ctrlO3}(x) : \emptyset \Rightarrow \{\mathrm{O3}(x)\}), (w : \emptyset \Rightarrow \emptyset)\}$
$\mathcal{T}_7 = \{$ $(e.0.0.0.0.0{:}\{\mathrm{L00}, \mathrm{I}(0), \mathrm{O1}(0), \mathrm{O2}(0), \mathrm{O3}(0)\} \Rightarrow \{\mathrm{L00}\}),$
$\ldots$
$(e.4.8.4.4.4{:}\{\mathrm{L}(4), \mathrm{I}(8), \mathrm{O1}(4), \mathrm{O2}(4), \mathrm{O3}(4)\} \Rightarrow \{\mathrm{L02}, \neg\mathrm{L00}\}),$
$\ldots$
$(e.99.1.0.0.0{:}\{\mathrm{L99}, \mathrm{I}(1), \mathrm{O1}(0), \mathrm{O2}(0), \mathrm{O3}(0)\} \Rightarrow \{\mathrm{L100}, \neg\mathrm{L99}\})$ $\}$

The threads $\mathcal{T}_4$ to $\mathcal{T}_6$ model the oil control activities of each agent. To simplify the exposition, we assume that any time an proposition is added we delete all other possible instantiantions of that proposition, e.g. for valve O1 writing an effect like $\{\mathrm{O1}(2)\}$ is a shortcut for $\{\mathrm{O1}(2), \neg\mathrm{O1}(0), \neg\mathrm{O1}(1), \neg\mathrm{O1}(3), \ldots, \neg\mathrm{O1}(9)\}$.

The threads $\mathcal{T}_1$ to $\mathcal{T}_3$ model the personal tasks of each agent, whose only hypothesis is that their effects do not modify the state of the valves, nor the level of the tank. The thread $\mathcal{T}_7$ models the evolution of the tank level. Notice that only the operators on this thread modify the level of the tank.

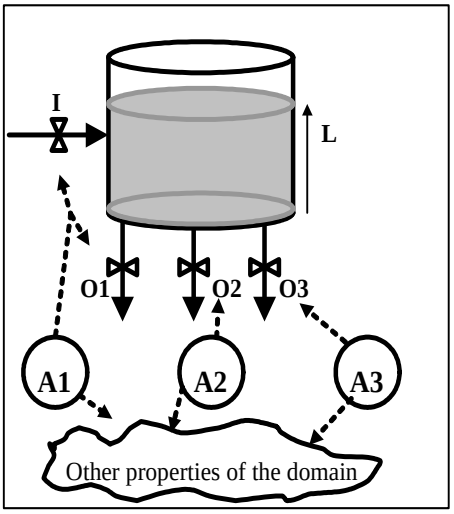

Fig. 1. The Shared Oil Tank Domain Example

The names of the operators of the tank level thread are mnemonic: each number represents the required precondition over the level of the tank, the input valve I flow, and the flow of each output valve O1, O2, and O3. By construction, we have that only one operator of the tank thread $\mathcal{T}_7$ is executable in any state of the world. Hence, if a plan is like:

$$\mathbb{P} \equiv \begin{bmatrix} \alpha_{1,i_1} & \alpha_{1,i_2} & \alpha_{1,i_3} & \alpha_{1,i_4} \\ \alpha_{2,i_1} & \alpha_{2,i_2} & \alpha_{2,i_3} & \alpha_{2,i_4} \\ \alpha_{3,i_1} & \alpha_{3,i_2} & \alpha_{3,i_3} & \alpha_{3,i_4} \\ \text{ctrlI}(7) & \text{ctrlO1}(3) & w & \text{ctrlO1}(3) \\ w & \text{ctrlO2}(2) & w & \text{ctrlO1}(3) \\ w & \text{ctrlO3}(5) & w & \text{ctrlO1}(3) \\ e.0.0.0.0.0 & e.0.7.0.0.0 & e.7.7.3.2.5 & e.4.7.3.2.5 \end{bmatrix}$$

It produces the following intermediate states of the shared oil tank [3]:

$$s_0 \models^{(\mathbb{P})_1} s_1 \models^{(\mathbb{P})_2} s_2 \models^{(\mathbb{P})_3} s_3 \models^{(\mathbb{P})_4} s_4$$

where: $s_0 = \{\text{I}(0), \text{O1}(0), \text{O2}(0), \text{O3}(0), \text{L00}\}$
$s_1 = \{\text{I}(7), \text{O1}(0), \text{O2}(0), \text{O3}(0), \text{L00}\}$
$s_2 = \{\text{I}(7), \text{O1}(3), \text{O2}(2), \text{O3}(5), \text{L07}\}$
$s_3 = \{\text{I}(7), \text{O1}(3), \text{O2}(2), \text{O3}(5), \text{L04}\}$
$s_4 = \{\text{I}(7), \text{O1}(3), \text{O2}(3), \text{O3}(3), \text{L02}\}$

After the first step, the plan fixes the input valve to a flow rate of 7, but this will affect only the level of the tank at the next step. Therefore, the agents cannot consume oil until the tank level raises; this will happen in s_2, after the second step is executed. The plan fixes the consumption rate of the three output valves O1, O2, and O3 respectively to 3, 2, and 5. Clearly this will not not affect the

[3] We abstract in the description of the states anything that the α operators have done to the other properties of the domain.

level until the next step. When executing the third step we have a previous level of L07, an input flow of 7 and the output valves flow fixed as commented in $3, 2$, and 5, hence, the new level is L04.

Notice that the tank level is treated as a *process*, (i.e. activities that evolve independently of the agent actions) that evolve following the physical rules of flow exchange. Additionally, this example shows that we can model this scenario successfully with CSTRIPS even without using numbers. The representation of concurrent *processes* in CSTRIPS is therefore quite immediate.

4 Worst Case Complexity of CSTRIPS

In what follows CSTRIPS(n) will denote the subset of CSTRIPS instances with exactly n threads. It is important to highlight that CSTRIPS with a single thread covers exactly the STRIPS semantics. We denote by I_{STRIPS} the set of all instances of the STRIPS planning problem and, for any $x = (\mathcal{D}, s_0, \mathcal{G}) \in I_{\text{STRIPS}}$, we denote $SOL_{\text{STRIPS}}(x)$ the set of all solutions of x. Therefore:

- $I_{\text{STRIPS}} = I_{\text{CSTRIPS(1)}}$
- $\forall x \in I_{\text{STRIPS}} \; . \; SOL_{\text{CSTRIPS(1)}}(x) = SOL_{\text{STRIPS}}(x)$

A 1-thread plan $\mathbb{P} \in SOL_{\text{CSTRIPS(1)}}$ is just a sequence of instantiated operators, exactly like in STRIPS. This means that we can interpret any STRIPS instance as a CSTRIPS(1) instance, and their solutions will be the same. STRIPS and CSTRIPS(1) are formally equivalent, and have the same expressivity and complexity.

Indeed, they differ for their ontological commitment. It is worth saying that the interpretation of a STRIPS instance as a CSTRIPS(1) instance goes against the ontological commitment made by CSTRIPS in any case that the STRIPS instance is modeling more than one concurrent thread. In these cases, to respect the ontological commitment, what should be done is to translate the STRIPS instance into the corresponding CSTRIPS(n) instance with n equal to the number of threads modeled by the STRIPS instance. As usual in STRIPS, the concurrent threads are modeled by means of the objects of the logic $\mathcal{L}$. For example, if a domain consists of n robots that in principle can move to different places in parallel, it is usual to model this with the operator $\text{Go}(x, y)$, where x represents the robot and y the place at which the robot x goes. Instead in CSTRIPS(n) we model this with one concurrent thread for each robot. Each concurrent thread will have an operator $\text{Go}(y)$, where y represents the place to which the robot of that thread goes.

Planning algorithms and complexity. Even if STRIPS is a particular case of CSTRIPS and, in a sense, CSTRIPS is more expressive than STRIPS because it allow us to explicitly express all the concurrent properties of a domain, it is worth noting that we have not increased the complexity of the planning problem for the expanded instances. This is an interesting property of CSTRIPS that we prove now.

Given any instance $x \in I_{\text{CSTRIPS}}$ we denote by $\mathcal{E}(x) = (\mathcal{E}(\mathcal{D}(x)), s_0(x), \mathcal{G})$ the **expansion of** x and by $\mathcal{E}(I_{\text{CSTRIPS}}) = \{\mathcal{E}(x) \mid x \in I_{\text{CSTRIPS}}\}$ the **set of**

all expanded instances of the CSTRIPS planning problem. Correspondingly, $\mathcal{E}(I_{\text{STRIPS}})$ for the set of all expanded instances of the STRIPS planning problem.

The total function iC2S : $\mathcal{E}(I_{\text{CSTRIPS}}) \rightarrow \mathcal{E}(I_{\text{STRIPS}})$, called the **instance-CSTRIPS-to-STRIPS translation**, is defined for any instance $x \in \mathcal{E}(I_{\text{CSTRIPS}})$ by: $\text{iC2S}(x) = (\mathcal{ED}_{\text{STRIPS}}, s_0(x), \mathcal{G}(x)) \in I_{\text{STRIPS}}$

with:

- $\mathcal{ED}_{\text{STRIPS}} = \{\, \alpha\theta \mid \forall \overrightarrow{\alpha\,\theta} \in \mathcal{E}(\mathcal{D}(x)) \wedge \alpha\theta = (\text{n}'_\alpha \,:\, \text{Pre}(\overrightarrow{\alpha\,\theta}) \Rightarrow \text{Eff}(\overrightarrow{\alpha\,\theta})\,)\,\}$
- $\text{n}'_\alpha = \mathcal{N}((\overrightarrow{\alpha\,\theta})_1)$ ";" $\mathcal{N}((\overrightarrow{\alpha\,\theta})_2)$ ";" ... ";" $\mathcal{N}((\overrightarrow{\alpha\,\theta})_n)$

This function converts any expanded instance of CSTRIPS into an expanded instance of STRIPS encoding in the name of each STRIP operator the names of the n components of the instantiated n-threads operator [4] $\overrightarrow{\alpha\,\theta}$. Having defined the state space in CSTRIPS exactly as the state space of CSTRIPS, we can use directly $s_0(x)$ and a reachability goal $\mathcal{G}(x)$ as the initial state and goal of the STRIPS instance. It is easy to see that iC2S can be computed in polynomial time.

pS2C : $\text{plans}_{\text{STRIPS}}(\mathcal{L}_{\mathcal{D}STRIPS}) \rightarrow \text{plans}(\mathcal{L}_{cD})$, called the **plans-STRIPS-to-CSTRIPS translation**, is the total function that converts any plan of a STRIPS instance that was "encoded" through iC2S of a CSTRIPS domain into the corresponding n-threads plan and leaves other STRIPS plans as instances of CSTRIPS(1). It is easy to give an algorithm that computes this function in polynomial time. For any $p \in SOL_{\text{STRIPS}}(x)$ it simply "parses" the name of each operator in p and constructs the corresponding instantiated n-threads operator. In general an "encoded" STRIPS operator has a name like $\text{n}_\alpha = n_1; n_2; \ldots; n_n$. Hence, for any n_i in the name n_α, the algorithm simply looks up in the corresponding *ith* thread of the CSTRIPS's domain $\mathcal{D}(x)$ the operator with that name. In that way it recovers the *ith* component $(\overrightarrow{\alpha\,\theta})_i)$ of the n-threads operator. This algorithm is also polynomial.

Notice that the defined total functions pS2C and iC2S guarantee us that the solutions are preserved:

$\forall x \in \mathcal{E}(I_{\text{CSTRIPS}})\ .\ \forall \mathbb{P} \in \text{pS2C}(SOL_{\text{STRIPS}}(\text{iC2S}(x))) \Leftrightarrow \mathbb{P} \in SOL_{\text{CSTRIPS}}(x)$

and being both functions computable by a polynomial time algorithm we have reduced the expanded CSTRIPS planning problem to the expanded STRIPS planning problem preserving the set of solutions. Also, we have shown before that $I_{\text{STRIPS}} \subseteq I_{\text{CSTRIPS}}$, hence, we have already proved the following:

Theorem 8 *The expanded CSTRIPS planning problem has the same complexity as the expanded STRIPS problem.*

Remark 9 *Any planning algorithm of STRIPS can be used for CSTRIPS.*

Though, in general a STRIPS planning algorithm is not optimal for CSTRIPS planning, because it does not take advantage of the explicit representation of concurrency.

[4] Without loss of generality we assume that ";" $\notin \mathcal{N}$ and that this concatenations n'_α are valid name symbols for STRIPS operators. Hence, ";" can be used to properly separate the n different names.

5 Taking Advantage of Explicit Concurrency

We now turn our attention to the potential gains of using explicit threads in CSTRIPS. The following remark sheds some light on this aspect.

Remark 10 *CSTRIPS* ***reduces the branching factor*** *when searching for a plan in concurrent domains because of its explicit representation of concurrency. It does this* ***by a factor of*** $\frac{1}{n^n}$ *when the instantiated operators are homogeneously distributed through the* n *concurrent threads.*

Let us suppose that we have a STRIPS domain $\mathcal{D}$ that models n concurrent threads, e.g. there are n robots that can perform one activity at the time, but that can operate in parallel in the domain. We denote with $\mathcal{E}(\mathcal{D})$ the set of all possible instantiated operators of $\mathcal{D}$ and assume that this set is finite and $|\mathcal{E}(\mathcal{D}(x))| = M$. The fact that the domain has n concurrent threads cannot be explicitly represented in STRIPS, and the effect of this is that when searching for a plan the planner needs to consider the possible application of all M instantiated operators. Hence, to consider one step of evolution of the n threads, e.g. one action per robot, we need to consider n times M instantiated operators, i.e. M^n possibilities.

When modeling the same problem in CSTRIPS we use an n concurrent threads domain. Therefore, the M possible actions are now divided into the n concurrent threads. We assume that this distribution is homogeneous, hence that each thread has $\frac{M}{n}$ possible actions. Therefore, to consider one step of evolution of the n threads, we need to consider n times but only $\frac{M}{n}$ possibilities, i.e. $(\frac{M}{n})^n$. Hence, in this case, we have $\frac{1}{n^n}$ less choices that in STRIPS [5].

The reader should not interpret that a CSTRIPS planner needs to work with the expanded domain $\mathcal{E}(\mathcal{D})$ of n-threadsoperators. We have used $\mathcal{E}(\mathcal{D})$ to characterize the semantics and complexity of CSTRIPS but, as shown here, in some domains the branching factor can be reduced and there are many ways of exploiting the explicit representation of concurrency.

6 Conclusions

This paper has formally introduced CSTRIPS, an explicit concurrent planning framework, by giving its syntax and semantics. CSTRIPS provides a theoretical reference, in particular for producing other explicit concurrent planning extensions for different planning frameworks. The paper also analyzes the relation between CSTRIPS and STRIPS, and proves that the computational complexity of the expanded CSTRIPS planning problem is the same as that of the expanded STRIPS planning problem. Hence, even if CSTRIPS is more expressive than STRIPS, in the sense that it allows us to explicitly model the concurrent activities of a domain, this increased expressiveness is not paid off by an increased complexity. We have also proved that any planning algorithm for STRIPS can be

[5] Notice that in CSTRIPS one step of evolution of the n threads corresponds to the application of just one $\overrightarrow{\alpha}\,\overrightarrow{\theta}$ operator, i.e. one step of the plan $\mathbb{P}$. In STRIPS this corresponds to n steps of the plan.

used for CSTRIPS, even if in general it will be suboptimal. In addition, the Shared Oil Tank example has shown several advantages of the explicit representation of concurrency. It shows how simple it is to model *processes* within CSTRIPS, even processes that are *resources* concurrently used by different agents. For a discussion about the difficulties encountered by others to model processes see [20]. The example also highlights the potential of CSTRIPS to structure the description of a domain by separating its different components.

A further remark is worth doing with respect to the differences between the present paper and [5] that, as said before, is the closest to ours. In [5] each operator needs to have by convention as its first argument a free variable denoting the acting agent (e.g. for an agent a and operator is like $\alpha(a, \overrightarrow{x})$), thus, assuming that each agent can perform only one action at a time, so any concurrent action must be perform by distinct agents. With CSTRIPS we have taken the representation of concurrent actions one step higher in abstraction and define a domain directly as a set of *concurrent threads*, each represented by a set of operators, thus, representing when applied a different thread of concurrent activity. Hence, operators do not need to have by convention the agent as first argument anymore and an agent can be easily modeled as having more than one concurrent thread as is shown in ours example. Clearly, by taking this step we have left the STRIPS framework. Therefore, it was necessary to deliver a formal semantic for our new framework. CSTRIPS also overcomes a pending issues noticed in [5]: the problem of which is the precise effect of concurrently executed actions that negate some precondition of another concurrent action. This is done by defining formally when concurrent actions (instantiated operators) are consistent and by introducing this as a condition for an operator to be executable. In conclusion, it is worth saying that our approach, even if exemplified here as an extension of STRIPS, can be straightforwardly applied to many other current planning frameworks like ADL [24] and PDDL [19].

Future Work. We are planning to present in another paper the treatment inside CSTRIPS of durative actions, extended goals and of three important features that are already treated in [5]: conditional effects; interaction of concurrent actions; and planning algorithms (an extension of a partial-order planner is presented in [5]). The first is a straight forward extension of CSTRIPS. With respect to the second we will present a specific approach completely different from the one presented in [5]. Finally, with respect to the third, even if we have shown that it is possible to use any STRIPS planning algorithm to find plans for any CSTRIPS planning problem, it is clear that this is the most inefficient way of planning in CSTRIPS. We are developing new planning algorithms that take advantage of this reduction and in general of the more structured representation of the concurrency of a domain specified in CSTRIPS.

Acknowledgements. We are grateful to Luigia Carlucci Aiello for all her time and effort. Amedeo Cesta's research has been partially supported by ASI (Italian Space Agency) under the basic research program (projects ARISCOM, DOVES and SACSO).

References

1. F. Bacchus. The power of modeling - a response to PDDL2.1 (commentary). *Journal of Artificial Intelligence Research*, 20:125–132, 2003.
2. F. Bacchus and M. Ady. Planning with resources and concurrency, a forward chaining approach. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 417–424, 2001.
3. Christer Bäckström. *Computational Complexity of Reasoning About Plans*. PhD thesis, Linköping University, 1992.
4. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-Based Scheduling. Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2001.
5. C. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.
6. M. Brenner. A formal model for planning with time and resources in concurrent domains. In *Proceedings of Workshop on Planning with Resources (IJCAI-01)*, Seattle, Washington, USA, August 2001. Morgan Kaufmann.
7. M. Brenner. Multiagent planning with partially ordered temporal plans. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003. Morgan Kaufmann.
8. C. Cheng and S.F. Smith. Generating feasible schedules under complex metric constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*, 1994.
9. G. De Giacomo, Y. Lesperance, and H.J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.
10. K. Erol. *Hierarchical Task Network Planning: Formalization, Analysis and Implementation*. PhD thesis, Univ. of Maryland, 1995.
11. M. Fox and D. Long. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003. Special issue on 3rd International Planning Competition.
12. J. Frank, K. Golden, and A. Jonsson. The loyal opposition comments on plan domain description languages. In *Proceedings of the Workshop on PDDL (ICAPS'03)*, Trento, Italy, June 2003.
13. J. Frank and A. Jonsson. Constraint based attribute and interval planning. *Journal of Constraints*, 8(4), 2003.
14. Patrice Godefroid and Froduald Kabanza. An efficient reactive planner for synthesizing reactive plans. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 640–645, Anaheim, July 1991.
15. D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173:1–39, 2002.
16. Amy L. Lansky. Localized planning with diverse plan construction methods. In *Artificial Intelligence*, volume (98)1-2, pages 49–136, January 1998.
17. T.L. Mc Cluskey. PDDL: A language with a purpose? In *Proceedings of the Workshop on PDDL (ICAPS'03)*, Trento, Italy, June 2003.
18. D. Mc Dermott and M.H. Burstein. Extending an estimated-regression planner for multi-agent planning. In *Proceedings AAAI Workshop on Planning by and for Multi-Agent Systems*, 2002.
19. Drew Mc Dermott. The 1998 AI planning systems competition. *AI Magazine*, 2:634–639, 1998.
20. Drew Mc Dermott. The formal semantics of processes in pddl. In *Proceedings of the Workshop on PDDL (ICAPS'03)*, Trento, Italy, June 2003.
21. Drew Mc Dermott, M. Ghallab, A. Howe, C.A. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003,DCS TR-1165, Yale Center for Communicational Vision and Control, October 1998.
22. N. Muscettola. HSTS: Integrating planning and scheduling. *In M.Zweben and M.S.Fox (Ed.)* Intelligent Scheduling, *Morgan Kauffmann*, 1994.
23. Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.
24. Edwin P.D. Pednault. *Toward a mathematical theory of plan synthesis*. PhD thesis, Stanford University, Department of Electrical Engineering, 1986.
25. Raymond Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press,Cambridge, Massachusetts, 2000.