DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No.:	DSIC **Pages:** 20
Title:	Instrumenting the CSP Semantics to Generate a Petri Net
Author(s):	M. Llorens, J. Oliver, J. Silva and S. Tamarit
Date:	February, 2010
Keywords:	Concurrent programming, Semantics, CSP, Petri nets

V° B°
Leader of research Group Author(s)

Instrumenting the CSP Semantics to Generate a Petri Net*

Marisa Llorens, Javier Oliver, Josep Silva, and Salvador Tamarit

Universidad Politécnica de Valencia, Camino de Vera S/N, E-46022 Valencia, Spain.
{mllorens,fjoliver,jsilva,stamarit}@dsic.upv.es

Abstract. Concurrent languages allow us to specify and simulate complex systems where many individual components run in parallel and communicate via message passing and synchronizations. However, these systems often pose additional difficulties to their comprehension and development due to the complexity imposed by the non-deterministic execution order of processes and to the restrictions imposed on this order by synchronizations. Two of the most extended used languages of this kind are CSP and Petri nets. Both models have been successfully used in the industry and many verification, simulation and analysis techniques exist for them. While CSP is very expressive and many techniques devoted to verify CSP specifications exist, Petri nets are specially interesting for simulation because they allow us to graphically animate specifications step by step. In this work, we formally define a transformation that allows us to automatically transform a CSP specification into an equivalent Petri net. This is the first approach that generates the Petri net as a side-effect of an instrumented semantics. The main advantage of this new approach is that the Petri net generated is very similar (structurally) to the CSP specification.

Keywords: Concurrent programming, Semantics, CSP, Petri nets.

1 Introduction

The increasing complexity of distributed and multiprocessor systems is making the industry to invest in concurrent languages that allow us to model many heterogeneous components able to interact in parallel and that can be automatically verified thanks to the development of modern techniques for the analysis and verification of such languages.

Two of the most important concurrent languages are the Communicating Sequential Processes (CSP) [5, 14] and the Petri nets [11, 13]. CSP is a very expressive process algebra with a big collection of tools for the specification and

* This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by the *Generalitat Valenciana* under grant ACOMP/2009/017, and by the *Universidad Politécnica de Valencia* (Programs PAID-05-08 and PAID-06-08). Salvador Tamarit was partially supported by the Spanish MICINN under FPI grant BES-2009-015019.

verification of complex systems. In fact, CSP is currently one of the most extended concurrent languages and it is being successfully used in many industrial projects. Complementarily, Petri nets are particularly useful for the simulation and animation of concurrent specifications. They can be used to graphically animate a specification and observe the synchronization of components step by step.

In this work we define a fully automatic transformation that allows us to transform a CSP specification into an equivalent Petri net (i.e., the sequences of observable events produced are exactly the same). This result is very interesting because it allows CSP developers not only to graphically animate their specifications through the use of the equivalent Petri net, but it also allows them to use all the tools and analysis techniques developed for Petri nets.

Our transformation is based on an instrumentation of the CSP's operational semantics. Roughly speaking, we define an algorithm that explores all computations of a CSP specification by using the instrumented semantics. The execution of the semantics produces as a side-effect the Petri net associated to each computation, and thus the final Petri net is produced incrementally.

The rest of the paper has been organized as follows. Section 2 overviews previous approaches to the transformation of CSP into Petri nets. In Section 3 we recall CSP and Petri nets. Section 4 presents an algorithm able to generate a Petri net equivalent to a given CSP specification. To obtain the Petri net, the algorithm uses an instrumentation of the standard operational semantics of CSP which is also introduced in this section. Finally, Section 5 concludes.

2 Related work

The transformation of CSP to Petri nets is an old objective of the community of concurrent programming languages. It not only has a clear practical utility, but it also has a wide theoretical interest because both concurrent models are very different, and establishing relations between them allows us to extend results from one model to the other. However, the problem of transforming a CSP specification into an equivalent Petri net is complex due to the big differences that exist between both formalisms. For this reason, many of the previous approaches have been criticized because it is hardly possible to see a relation between the generated Petri net and the CSP specification (i.e., when a transition of the Petri net is fired, it is not even clear to what CSP process corresponds this transition). In this respect, the transformation presented here is particularly interesting because the Petri net is generated directly from the operational semantics in such a way that each syntactic element of the CSP specification has a representation in the Petri net. Hence, it is very easy to map the animation of the Petri net to the CSP specification.

There are two major research lines aimed at transforming CSP to Petri nets. The first line is based on traces describing the behaviour of the system. In [9], starting from a trace-based representation of the behaviour of the system, according to a subset of the Hoare's theory where no sequential composition with

recursion is allowed, a Stochastic Petri net model is built in a modular and systematic way. The overall model is built by modelling the system's components individually, and then putting them together by means of superposition. The second line of research includes all methodologies that translate CSP specifications into Petri nets. One of the first works translating CSP to Petri nets was [1], where distributed termination is assumed but nesting of parallel commands is not allowed. In [3], a CSP-like language is considered and translated into a subclass of Pr/T nets with individual tokens, where neither nesting of parallel commands is allowed nor distributed termination is taken into account. Other papers in this area are [12] that considers a subset of CCSP (the union of Milner's CCS[10] and Hoare's CSP[5]), and [2] which provides full CSP with a truly concurrent and distributed operational semantics based on C/E Systems. There are also some works [15, 8] that translate the CSP specifications into Stochastic or Timed Petri nets in order to perform real-time analysis and performance evaluation. As in our work, all these papers do not allow recursion of nested parallel processes because the set of places of the generated Petri net would be infinite. In some way, our new semantics-based approach opens a third line of research where the transformation is directed by the semantics.

3 CSP and Petri nets

3.1 The syntax and semantics of CSP

In order to make the paper self-contained, this section recalls CSP's syntax and semantics. For concretion, and to facilitate the understanding of the following definitions and algorithm, we have selected a subset of CSP that is sufficiently expressive to illustrate the method, and it contains the most important operators that produce the challenging problems such as deadlocks, non-determinism and parallel execution.

Domains :	$M, N \ldots \in Names$ (Process names)	$P, Q \ldots \in Procs$ (Processes)	$a, b \ldots \in \Sigma$ (Events)
	$S ::= D_1 \ldots D_m$ (Entire specification)		
	$D ::= N = P$ (Process definition)		
	$P ::= M \mid a \rightarrow P \mid P \sqcap Q \mid P \,\Box\, Q \mid P \underset{X \subseteq \Sigma}{\parallel} Q \mid STOP$		

Fig. 1. Syntax of CSP specifications

Figure 1 summarizes the syntax constructions used in CSP [5] specifications. A *specification* is a finite collection of definitions. The left-hand side of each definition is the name of a different process, which is defined in the right-hand side (abbrev. *rhs*) by means of an expression that can be a call to another process or a combination of the following operators:

Prefixing $(a \rightarrow P)$ Event a must happen before process P.

Internal choice ($P \sqcap Q$) The system (e.g., non-deterministically) chooses to execute one of the two processes P or Q.
External choice ($P \Box Q$) It is identic to internal choice but the choice comes from outside the system (e.g., the user).
Synchronized parallelism ($P \underset{X \subseteq \Sigma}{||} Q$) Both processes are executed in parallel with a set X of synchronized events. In absence of synchronizations both processes can execute in any order. Whenever a synchronized event $a \in X$ happens in one of the processes, it must also happen in the other at the same time. Whenever the set of synchronized events is not specified, it is assumed that processes are synchronized in all common events. A particular case of parallel execution is *interleaving* (represented by $|||$) where no synchronizations exist (i.e., $X = \emptyset$).
Stop ($STOP$) Synonym of deadlock: It finishes the current process.

Example 1. The following simple CSP specification represents an astronaut that gets a medal from NASA if she succeeds in a space mission:

$$\begin{array}{l}
\texttt{MAIN} = (\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{||} \texttt{NASA}) \\
\texttt{ASTRONAUT} = \texttt{mission} \rightarrow \texttt{MISSION} \\
\texttt{MISSION} = (\texttt{fail} \rightarrow \texttt{STOP}) \Box (\texttt{success} \rightarrow \texttt{STOP}) \\
\texttt{NASA} = \texttt{success} \rightarrow \texttt{medal} \rightarrow \texttt{STOP}
\end{array}$$

We now recall the standard operational semantics of CSP as defined by Roscoe [14]. It is presented in Fig. 2 as a logical inference system. A *state* of the semantics is a process to be evaluated called the *control*. In the following, wlog we assume that the system starts with an initial state `MAIN`, and the rules of the semantics are used to infer how this state evolves. When no rules can be applied to the current state, the computation finishes. The rules of the semantics change the states of the computation due to the occurrence of events. The set of possible events is $\Sigma^\tau = \Sigma \cup \{\tau\}$. Events in Σ are visible from the external environment, and can only happen with its co-operation (e.g., actions of the user). Event τ is an internal event that cannot be observed from outside the system and it happens automatically as defined by the semantics.

In order to perform computations, we construct an initial state and (non-deterministically) apply the rules of Fig. 2. Their intuitive meaning is the following: (Process Call) The call to process N is unfolded and $rhs(N)$ is added to the control.
(Prefixing) When event a occurs, process P is added to the control.
(Internal Choice 1 and 2) The system (non-deterministically), with the occurrence of τ, selects one of the two processes P or Q which is added to the control.
(External Choice 1, 2, 3 and 4) The occurrence of τ develops one of the processes. The occurrence of an event $e \in \Sigma$ is used to select one of the two processes P or Q and the control changes according to the event.
(Synchronized Parallelism 1 and 2) When a non-synchronized event happens, one of the two processes P or Q evolves accordingly.

(Process Call) $$\frac{}{N \xrightarrow{\tau} rhs(N)}$$

(Prefixing) $$\frac{}{(a \to P) \xrightarrow{a} P}$$

(Internal Choice 1) $$\frac{}{(P \sqcap Q) \xrightarrow{\tau} P}$$

(Internal Choice 2) $$\frac{}{(P \sqcap Q) \xrightarrow{\tau} Q}$$

(External Choice 1) $$\frac{P \xrightarrow{\tau} P'}{(P \;\Box\; Q) \xrightarrow{\tau} (P' \;\Box\; Q)}$$

(External Choice 2) $$\frac{Q \xrightarrow{\tau} Q'}{(P \;\Box\; Q) \xrightarrow{\tau} (P \;\Box\; Q')}$$

(External Choice 3) $$\frac{P \xrightarrow{e} P'}{(P \;\Box\; Q) \xrightarrow{e} P'} \; e \in \Sigma$$

(External Choice 4) $$\frac{Q \xrightarrow{e} Q'}{(P \;\Box\; Q) \xrightarrow{e} Q'} \; e \in \Sigma$$

(Synchronized Parallelism 1) $$\frac{P \xrightarrow{e} P'}{(P \underset{X}{||} Q) \xrightarrow{e} (P' \underset{X}{||} Q)} \; e \in \Sigma^{\tau} \backslash X$$

(Synchronized Parallelism 2) $$\frac{Q \xrightarrow{e} Q'}{(P \underset{X}{||} Q) \xrightarrow{e} (P \underset{X}{||} Q')} \; e \in \Sigma^{\tau} \backslash X$$

(Synchronized Parallelism 3) $$\frac{P \xrightarrow{e} P' \quad Q \xrightarrow{e} Q'}{(P \underset{X}{||} Q) \xrightarrow{e} (P' \underset{X}{||} Q')} \; e \in X$$

Fig. 2. CSP's operational semantics

(Synchronized Parallelism 3) When a synchronized event ($e \in X$) happens, it is required that both processes synchronize; P and Q are executed at the same time and the control becomes $P' \underset{X}{||} Q'$.

3.2 Labeled Petri nets

In this section, we recall some basic concepts of Petri nets needed along the paper:

Definition 1. *(Petri Net) A* Petri net *[11, 13] is a tuple $N = (P, T, F)$, where P is a finite set of* places, *T is a finite set of* transitions, *such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$ and F is a finite set of weighted* arcs *representing the* flow relation *$F : P \times T \;\cup\; T \times P \to \mathbb{N}$. A* marking *of a Petri net is a function $M : P \to \mathbb{N}$. A* marked Petri net *is a pair (N, M_0) where M_0 is a marking of the Petri net called an* initial marking.

Definition 2. *(Labeled Petri Net) A* labeled Petri net *[4, 11, 13] is a 6-tuple $\mathcal{N} = (\langle P, T, F \rangle, M_0, \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T)$, where $\langle P, T, F \rangle$ is a Petri net, M_0 is the* initial marking, *$\mathcal{P}$ is a* place alphabet, *$\mathcal{T}$ is a* transition alphabet, *$\mathcal{L}_P$ is a* labeling function *$\mathcal{L}_P : P \to \mathcal{P}$ and $\mathcal{L}_T$ is a* labeling function *$\mathcal{L}_T : T \to \mathcal{T}$.*

In the following, we will use *labeled ordinary Petri nets* where all of its arc weights are 1, $\mathcal{L}_P$ is a partial function and $\mathcal{L}_T$ is a total function. The notion of firing sequence is used in the paper according to the generally accepted definition [4, 11, 13]. For the sake of concreteness, we often use the notation p_α (t_β) to denote the place p (transition t) whose label is $\alpha \in \mathcal{P}$ ($\beta \in \mathcal{T}$), i.e., $\mathcal{L}_P(p) = \alpha$ ($\mathcal{L}_T(t) = \beta$); or also to assign label α (β) to place p (transition t).

An example of Petri net is drawn in Fig. 3.

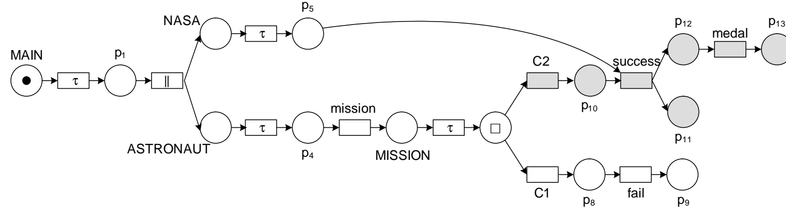

Fig. 3. PN associated to the specification of Example 1

4 An algorithm to transform CSP to Petri nets

This section introduces an algorithm able to generate a Petri net that produces the same sequences[1] of visible events as a given CSP specification. The algorithm uses an instrumented operational semantics of CSP which (i) generates as a side-effect a Petri net associated to the computation performed with the semantics; (ii) it controls that no infinite loops are executed; and (iii) it ensures that the execution is deterministic.

Algorithm 1 controls that the semantics is executed repeatedly in order to deterministically execute all possible computations—of the original (non-deterministic) specification—and the Petri net is constructed incrementally with each execution of the semantics. The key point of the algorithm is the use of a stack that records the actions that can be performed by the semantics. In particular, the stack contains tuples of the form $(rule, rules)$ where $rule$ indicates the rule that must be selected by the semantics in the next execution step, and $rules$ is a set with the other possible rules that can be selected. The algorithm uses the stack to prepare each execution of the semantics indicating the rules that must be applied at each step. For this, function `UpdStack` is used; it basically avoids to repeat the same computation with the semantics. When the semantics finishes, the algorithm prepares a new execution of the semantics with an updated stack. This is repeated until all possible computations are explored (i.e., until the stack is empty).

The semantics in Fig. 2 can be non-terminating due to infinite computations. Therefore, the instrumentation of the semantics incorporates a loop-checking mechanism to ensure termination.

The instrumented semantics used by Algorithm 1 is shown in Fig. 4. It is an operational semantics where a *state* is a tuple $(P, p, \mathcal{N}, C, (S, S_0), \Delta)$, where P is the process to be evaluated (the *control*), p is the last place added to the Petri net $\mathcal{N}$, C is the set of already performed process calls and it is used to avoid infinite unfolding of the same process. (S, S_0) is a tuple with two stacks (where the empty stack is denoted by []) that contains the rules to apply and the rules applied so far, and Δ is a set of references used to draw synchronizations in $\mathcal{N}$. The basic idea of the Petri net construction is to generate the Petri net associated

[1] In CSP terminology, these sequences are the so-called traces (see, e.g., chapter 8.2 of [14]). In Petri nets they correspond to transition firing sequences (see, e.g., [11]).

Algorithm 1 General Algorithm

Build the initial state of the semantics: $state = (\texttt{MAIN}, p_0, \mathcal{N}, \emptyset, ([], []), \emptyset)$
where $\mathcal{N} = (\langle\{p_0\}, \emptyset, \emptyset\rangle, M_0, \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T)$, $M_0(p_0) = 1$, $\mathcal{P} = Names \cup \{\Box\}$,
$\mathcal{T} = \Sigma^\tau \cup \{\|, \texttt{C1}, \texttt{C2}\}$.
repeat
  repeat
    Run the rules of the semantics with the state *state*
  until no more rules can be applied
  Get the new state $state = (_, _, \mathcal{N}, _, ([], S_0), _)$
  $state = (\texttt{MAIN}, p_0, \mathcal{N}, \emptyset, (\texttt{UpdStack}(S_0), []), \emptyset)$
until $\texttt{UpdStack}(S_0) = []$
return $\mathcal{N}$
where function `UpdStack` is defined as follows:

$$\texttt{UpdStack}(S) = \begin{cases} (rule, rules\backslash\{rule\}) : S' & \text{if } S = (_, rules) : S' \text{ and } rule \in rules \\ \texttt{UpdStack}(S') & \text{if } S = (_, \emptyset) : S' \\ [] & \text{if } S = [] \end{cases}$$

to the current control and connect this net to the last place added to $\mathcal{N}$. The set C is called the *context*. It is a collection of the already evaluated process calls and is used as a loop-checking mechanism to avoid infinite unfolding. C is a set of pairs (P', p') where P' is a process name that is represented in the Petri net with the place p'.

Given a labeled Petri net $\mathcal{N} = (\langle P, T, F\rangle, M, \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T)$ and the current reference $p \in P$, we use the notation $\mathcal{N}[p \mapsto t_a \mapsto p']$ either as a condition on $\mathcal{N}$ (i.e., $\mathcal{N}$ contains transition t_a), or also to introduce a transition t and a place p' into $\mathcal{N}$ producing the net $\mathcal{N}' = (\langle P', T', F'\rangle, M', \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T)$ where $P' = P \cup \{p'\}$, $T' = T \cup \{t_a\}$, $F' = F \cup \{(p, t_a), (t_a, p')\}$, $\forall p \in P : M'(p) = M(p) \wedge M'(p') = 0$ and $\mathcal{L}_T(t_a) = a$ where $a \in \mathcal{T}$.

An explanation for each rule of the semantics follows:
(Process Call) This rule basically decides whether process P must be unfolded or not. This is done with function `LoopCheck`. If the process has been previously unfolded (thus, C must contain a pair $(P, _)$), then we are in a loop, and P is marked as a loop with the special symbol $\circlearrowleft$. This label is later used by rule (Synchronized Parallelism 4) to decide whether the process should be unfolded again. If P has not been previously unfolded, then $rhs(P)$ becomes the new control. Observe that the new Petri net $\mathcal{N}'$ contains a place p_P that represents the process call and a transition t_τ that represents the occurrence of event τ. No event can synchronize in this rule, thus Δ is empty.

Function `LoopCheck`, used to prevent infinite unfolding, is defined below.

$$\texttt{LoopCheck}(P, p, \mathcal{N}, C) = \begin{cases} (\circlearrowleft(P), p'', \mathcal{N}' \cup \{t \mapsto p' \mid (P, p') \in C \wedge t \mapsto p \in \mathcal{N}\}) & \text{if } \exists (P, _) \in C \\ (rhs(P), p'', \mathcal{N}') & \text{otherwise} \end{cases}$$

where $\mathcal{N}' = \mathcal{N}[p_P \mapsto t_\tau \mapsto p'']$.

(Process Call)

$$\frac{}{(P, p, \mathcal{N}, C, (S, S_0), _) \xrightarrow{\tau} (P', p', \mathcal{N}', \{(P, p')\} \cup C, (S, S_0), \emptyset)}$$

$(P', p', \mathcal{N}') \ = \ \texttt{LoopCheck}(P, p, \mathcal{N}, C)$

(Prefixing)

$$\frac{}{(a \rightarrow P, p, \mathcal{N}, C, (S, S_0), _) \xrightarrow{a} (P, p', \mathcal{N}[p \mapsto t_a \mapsto p'], C, (S, S_0), \{(p, t_a, p')\})}$$

(Choice)

$$\frac{}{(P \Box Q, p, \mathcal{N}, C, (S, S_0), _) \xrightarrow{\tau} (P', p', \mathcal{N}', C, (S', S'_0), \emptyset)}$$

$(P', p', \mathcal{N}', (S', S'_0)) \ = \ \texttt{SelectBranch}(P \Box Q, p, \mathcal{N}, (S, S_0))$

(Synchronized Parallelism 1)

$$\frac{(P1, p'_1, \mathcal{N}', C'_1, (S', (\mathsf{SP1}, rules) : S_0), _) \xrightarrow{e} (P', p''_1, \mathcal{N}'', C''_1, (S'', S'_0), \Delta)}{(P1 \underset{X}{\|}_{lab} P2, p, \mathcal{N}, C, (S' : (\mathsf{SP1}, rules), S_0), _) \xrightarrow{e} (P' \underset{X}{\|}_{lab'} P2, p, \mathcal{N}'', C, (S'', S'_0), \Delta)} \quad e \in \Sigma^\tau \backslash X$$

$(\mathcal{N}', p'_1, C'_1, p'_2, C'_2) \ = \ \texttt{InitBranches}(\mathcal{N}, p_1, C_1, p_2, C_2, p, C) \wedge$

$lab = (p_1, C_1, p_2, C_2, \Upsilon) \wedge lab' = (p''_1, C''_1, p'_2, C'_2, \Upsilon)$

(Synchronized Parallelism 2)

$$\frac{(P2, p'_2, \mathcal{N}', C'_2, (S', (\mathsf{SP2}, rules) : S_0), _) \xrightarrow{e} (P', p''_2, \mathcal{N}'', C''_2, (S'', S'_0), \Delta)}{(P1 \underset{X}{\|}_{lab} P2, p, \mathcal{N}, C, (S' : (\mathsf{SP2}, rules), S_0), _) \xrightarrow{e} (P1 \underset{X}{\|}_{lab'} P', p, \mathcal{N}'', C, (S'', S'_0), \Delta)} \quad e \in \Sigma^\tau \backslash X$$

$(\mathcal{N}', p'_1, C'_1, p'_2, C'_2) \ = \ \texttt{InitBranches}(\mathcal{N}, p_1, C_1, p_2, C_2, p, C) \wedge$

$lab = (p_1, C_1, p_2, C_2, \Upsilon) \wedge lab' = (p'_1, C'_1, p''_2, C''_2, \Upsilon)$

(Synchronized Parallelism 3)

$$\frac{Left \quad Right}{(P1 \underset{X}{\|}_{lab} P2, p, \mathcal{N}, C, (S' : (\mathsf{SP3}, rules), S_0), _) \xrightarrow{e} (P1' \underset{X}{\|}_{lab'} P2', p, \mathcal{N}_s, C, (S''', S''_0), \Delta)} \quad e \in X$$

$(\mathcal{N}', p'_1, C'_1, p'_2, C'_2) \ = \ \texttt{InitBranches}(\mathcal{N}, p_1, C_1, p_2, C_2, p, C) \ \wedge$

$lab = (p_1, C_1, p_2, C_2, \Upsilon) \wedge lab' = (p''_1, C''_1, p''_2, C''_2, \bullet) \ \wedge$

$Left = (P1, p'_1, \mathcal{N}', C'_1, (S', (\mathsf{SP3}, rules) : S_0), _) \xrightarrow{e} (P1', p''_1, \mathcal{N}'', C''_1, (S'', S'_0), \Delta_1) \ \wedge$

$Right = (P2, p'_2, \mathcal{N}'', C'_2, (S'', S'_0), _) \xrightarrow{e} (P2', p''_2, \mathcal{N}''', C''_2, (S''', S''_0), \Delta_2) \ \wedge$

$\mathcal{N}_s = (\mathcal{N}''' \cup \{(p \mapsto t_e \mapsto p') \mid (p, _, p') \in (\Delta_1 \cup \Delta_2)\}) \backslash \{(p \mapsto t \mapsto p') \mid (p, t, p') \in (\Delta_1 \cup \Delta_2)\} \ \wedge$

$\Delta = \{(p, t_e, p') \mid (p, _, p') \in (\Delta_1 \cup \Delta_2)\}$

(Synchronized Parallelism 4)

$$\frac{}{(P1 \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P2, p, \mathcal{N}, C, (S' : (\mathsf{SP4}, rules), S_0), _) \xrightarrow{\tau} (P', p, \mathcal{N}', C, (S', (\mathsf{SP4}, rules) : S_0), \emptyset)}$$

$(P', \mathcal{N}') = \texttt{LoopControl}(P1 \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P2, \mathcal{N})$

(Synchronized Parallelism 5)

$$\frac{(P1 \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P2, p, \mathcal{N}, C, ([(rule, rules)], S_0), _) \xrightarrow{e} (P, p, \mathcal{N}', C, (S', S'_0), \Delta)}{(P1 \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P2, p, \mathcal{N}, C, ([], S_0), _) \xrightarrow{e} (P, p, \mathcal{N}', C, (S', S'_0), \Delta)} \quad e \in \Sigma^\tau$$

$rule \in \texttt{AppRules}(P1 \underset{X}{\|} P2) \wedge rules = \texttt{AppRules}(P1 \underset{X}{\|} P2) \backslash \{rule\}$

Fig. 4. An instrumented operational semantics that generates a Petri net

(Prefixing) This rule adds to $\mathcal{N}$ a transition t_a that represents the occurrence of event a. t_a is connected to the current place p and to a new place p'. The new

control is P. The new set Δ contains the tuple (p, t_a, p') to indicate that event a has occurred and it must be synchronized when required by (Synch. Parallelism 3).

(Choice) The only sources of non-determinism are choice operators (different branches can be selected for execution) and parallel operators (different order of branches can be selected for execution). Therefore, every time the semantics executes a choice or a parallelism, they are made deterministic thanks to the information in the stack S. In the case of choices, both internal and external can be treated with a single rule. No event can synchronize in this rule, thus Δ is empty. Function `SelectBranch` is used to produce the new control P' and the new tuple of stacks (S', S'_0), by selecting a branch with the information of the stack. Note that, for simplicity, the lists constructor ":" has been overloaded, and it is also used to build lists of the form $(A : a)$ where A is a list and a is the last element:

$$\texttt{SelectBranch}(P\Box Q, p, \mathcal{N}, (S, S_0)) \ = \\ \begin{cases} (P, p', \mathcal{N}[p_\Box \mapsto t_{C1} \mapsto p'], (S', (\mathsf{C1}, \{\mathsf{C2}\}) : S_0)) & \text{if } S = S' : (\mathsf{C1}, \{\mathsf{C2}\}) \\ (Q, p', \mathcal{N}[p_\Box \mapsto t_{C2} \mapsto p'], (S', (\mathsf{C2}, \emptyset) : S_0)) & \text{if } S = S' : (\mathsf{C2}, \emptyset) \\ (P, p', \mathcal{N}[p_\Box \mapsto t_{C1} \mapsto p'], ([], (\mathsf{C1}, \{\mathsf{C2}\}) : S_0)) & \text{otherwise} \end{cases}$$

If the last element of the stack S indicates that the first branch of the choice (C1) must be selected, then P is the new control. If the second branch must be selected (C2), the new control is Q. In any other case the stack is empty, and thus this is the first time that this choice is evaluated. Then, we select the first branch (P is the new control) and we add $(\mathsf{C1}, \{\mathsf{C2}\})$ to the stack S_0 indicating that C1 has been chosen, and the remaining option is C2. This function creates a new transition for each branch (t_{C1} and t_{C2}) that represents the τ event.

(Synchronized Parallelism 1 and 2) The stack determines what rule to use when a parallelism operator is in the control. If the last element in the stack is SP1, then (Synchronized Parallelism 1) is used. If it is SP2, (Synchronized Parallelism 2) is used.

In a parallelism, both parallel processes can be intertwiningly executed until a synchronized event is found. Therefore, places and transitions for both processes can be added interweaved to the Petri net. Hence, the semantics needs to know in every state the references to be used in both branches. This is done by labeling parallelism operators with a tuple of the form $(p_1, C_1, p_2, C_2, \Upsilon)$ where p_1 and p_2 are respectively the last places added to the left and right branches of the parallelism. C_1 and C_2 are the current contexts of the left and right branches of the parallelism; and Υ is a place of the Petri net used to decide when to unfold a process call (in order to avoid infinite loops). This tuple is initialized to $(\bullet, \emptyset, \bullet, \emptyset, \bullet)$ for every parallelism that is introduced in the computation. The set Δ is passed down unchanged so that another rule can use it if necessary.

These rules develop the branches of the parallelism until they are finished or until they must synchronize. They use function `InitBranches` to introduce the parallelism into the Petri net the first time it is executed and only if it has not been introduced in a previous computation.

After executing function `InitBranches`, we get a new graph and new references and contexts for each branch.

$$\texttt{InitBranches}(\mathcal{N}, p_1, C_1, p_2, C_2, p, C) \;=\; \begin{cases} (\mathcal{N}[p \mapsto t_{\|} \mapsto p_1', p \mapsto t_{\|} \mapsto p_2'], p_1', C, p_2', C) & \text{if } p_1 = \bullet \\ (\mathcal{N}, p_1, C_1, p_2, C_2) & \text{otherwise} \end{cases}$$

Observe that the parallelism operator is represented in the Petri net with a transition $t_{\|}$. This transition is connected to two new places (p_1' and p_2'), one for each branch.

(Synchronized Parallelism 3) It is applied when the last element in the stack is SP3. It is used to synchronize the parallel processes. In this rule, Υ is replaced by $\bullet$, meaning that a synchronization edge has been drawn and the loops could be unfolded again if it is needed. All the events that have been executed in this step must be synchronized. Therefore, all the events occurred in the subderivations of $P1$ (Δ_1) and $P2$ (Δ_2) are mutually synchronized. Note that this is done in the Petri net by removing the transitions that were added in each subderivation ($\{(p \mapsto t \mapsto p') \mid (p, t, p') \in (\Delta_1 \cup \Delta_2)\}$) and connecting all of them with a single transition t_e.

(Synchronized Parallelism 4) This rule is applied when the last element in the stack is SP4. It is used when none of the parallel processes can proceed (because they already finished, deadlocked or were labeled with $\circlearrowleft$). When a process is labeled as a loop with $\circlearrowleft$, it can be unlabeled to unfold it once[2] in order to allow the other processes to continue. This happens when the looped process is in parallel with other process and the later is waiting to synchronize with the former. In order to perform the synchronization, both processes must continue, thus the loop is unlabeled. This task is done by function `LoopControl`. It decides whether the branches of the parallelism should be further unfolded or they should be stopped (e.g., due to a deadlock or an infinite loop):

$$\texttt{LoopControl}(P \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} Q, \mathcal{N}) =$$

$$\begin{cases} (\circlearrowleft(P'_{\circlearrowleft} \underset{X}{\|}_{(p_1', C_1', p_2', C_2', \bullet)} Q'_{\circlearrowleft}), \mathcal{N}) & \text{if } P' = \circlearrowleft(P'_{\circlearrowleft}) \wedge Q' = \circlearrowleft(Q'_{\circlearrowleft}) \\ (\circlearrowleft(P'_{\circlearrowleft} \underset{X}{\|}_{(p_1', C_1', p_2', C_2', \bullet)} \texttt{STOP}), \mathcal{N}) & \text{if } P' = \circlearrowleft(P'_{\circlearrowleft}) \wedge (Q' = \texttt{STOP} \vee (Q' \neq \circlearrowleft(_) \vee \Upsilon = p_2')) \\ (P''_{\circlearrowleft} \underset{X}{\|}_{(p_1', C_1', p_2', C_2', p_2')} Q', \mathcal{N}') & \text{if } P' = \circlearrowleft(P'_{\circlearrowleft}) \wedge Q' \neq \texttt{STOP} \wedge Q' \neq \circlearrowleft(_) \wedge \Upsilon \neq p_2' \\ & \quad \wedge \; (P''_{\circlearrowleft}, \mathcal{N}') = \texttt{DelEdges}(P'_{\circlearrowleft}, \mathcal{N}, p_1') \\ (\texttt{STOP}, \mathcal{N}) & \text{otherwise} \end{cases}$$

where $(P', p_1', C_1', Q', p_2', C_2') \in \{(P, p_1, C_1, Q, p_2, C_2), (Q, p_2, C_2, P, p_1, C_1)\}$.

$$\texttt{DelEdges}(P, \mathcal{N}, p) =$$

$$\begin{cases} (P_1' \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P_2', \mathcal{N}'') & \text{if } P = P_1 \underset{X}{\|}_{(p_1, C_1, p_2, C_2, \Upsilon)} P_2 \wedge \\ & \quad (P_1', \mathcal{N}') = \texttt{DelEdges}(P_1, \mathcal{N}, p_1) \wedge (P_2', \mathcal{N}'') = \texttt{DelEdges}(P_2, \mathcal{N}', p_2) \\ (rhs(P), \mathcal{N} \backslash \{t \mapsto p'' \in \mathcal{N} | t \mapsto p' \mapsto t' \mapsto p \in \mathcal{N} \wedge p'' \neq p'\}) & \text{otherwise} \end{cases}$$

[2] Only once because it will be labeled again by rule (Process Call) when the loop is repeated.

When one of the branches has been labeled as a loop, there are three options: (i) The other branch is also a loop. In this case, the whole parallelism is marked as a loop, and Υ is put to $\bullet$. (ii) Either it is a loop that has been unfolded without drawing any synchronization (this is known because Υ is equal to the reference of the other branch), or the other branch already terminated (i.e., it is STOP). In this case, the parallelism is also marked as a loop, and the other branch is put to STOP (this means that this process has been deadlocked). Also here, Υ is put to $\bullet$. (iii) If we are not in a loop, then we allow the parallelism to proceed by unlabeling the looped branch. Here, function DelEdges is used to unfold the looped process calls, and to remove the edges introduced by rule (Process Call) that connect those process calls that were looped. In the rest of the cases STOP is returned representing that this is a deadlock, and thus, stopping further computations.
(Synchronized Parallelism 5) This rule is used when the stack is empty. It basically analyzes the control and decides what are the applicable rules of the semantics. This is done with function AppRules which returns the set of rules R that can be applied to a synchronized parallelism $P \underset{X}{\|} Q$:

$$\mathtt{AppRules}(P \underset{X}{\|} Q) = \begin{cases} \{\mathsf{SP1}\} & \text{if } \tau \in \mathtt{FstEvs}(P) \\ \{\mathsf{SP2}\} & \text{if } \tau \notin \mathtt{FstEvs}(P) \wedge \tau \in \mathtt{FstEvs}(Q) \\ R & \text{if } \tau \notin \mathtt{FstEvs}(P) \wedge \tau \notin \mathtt{FstEvs}(Q) \ \wedge \ R \neq \emptyset \\ \{\mathsf{SP4}\} & \text{otherwise} \end{cases}$$

$$\text{where} \begin{cases} \mathsf{SP1} \in R & \text{if } \exists e \in \mathtt{FstEvs}(P) \wedge e \notin X \\ \mathsf{SP2} \in R & \text{if } \exists e \in \mathtt{FstEvs}(Q) \wedge e \notin X \\ \mathsf{SP3} \in R & \text{if } \exists e \in \mathtt{FstEvs}(P) \wedge \exists e \in \mathtt{FstEvs}(Q) \wedge e \in X \end{cases}$$

Essentially, AppRules decides what rules are applicable depending on the events that could happen in the next step. These events can be inferred by using function FstEvs. In particular, given a process P, function FstEvs returns the set of events that can trigger a rule in the semantics using P as the control.

$$\mathtt{FstEvs}(P) =$$
$$\begin{cases} \{a\} & \text{if } \ P = a \rightarrow Q \\ \emptyset & \text{if } \ P = \circlearrowleft Q \ \vee \ P = \mathtt{STOP} \\ \{\tau\} & \text{if } \ P = M \ \vee \ P = Q \Box R \ \vee \ P = (\mathtt{STOP} \underset{X}{\|} \mathtt{STOP}) \\ & \vee \ P = (\circlearrowleft Q \underset{X}{\|} \circlearrowleft R) \ \vee \ P = (\circlearrowleft Q \underset{X}{\|} \mathtt{STOP}) \ \vee \ P = (\mathtt{STOP} \underset{X}{\|} \circlearrowleft R) \\ & \vee (P = (\circlearrowleft Q \underset{X}{\|} R) \wedge \mathtt{FstEvs}(R) \subseteq X) \vee (P = (Q \underset{X}{\|} \circlearrowleft R) \wedge \mathtt{FstEvs}(Q) \subseteq X) \\ & \vee (P = Q \underset{X}{\|} R \wedge \mathtt{FstEvs}(Q) \subseteq X \wedge \mathtt{FstEvs}(R) \subseteq X \wedge \bigcap_{M \in \{Q,R\}} \mathtt{FstEvs}(M) = \emptyset) \\ E & \text{otherwise, where } P = Q \underset{X}{\|} R \ \wedge \\ & E = (\mathtt{FstEvs}(Q) \cup \mathtt{FstEvs}(R)) \backslash \{e \mid e \in X \wedge \\ & ((e \in \mathtt{FstEvs}(Q) \wedge e \notin \mathtt{FstEvs}(R)) \vee (e \notin \mathtt{FstEvs}(Q) \wedge e \in \mathtt{FstEvs}(R))\} \end{cases}$$

Therefore, rule (Synchronized Parallelism 5) prepares the stack allowing the semantics to proceed with the correct rule.

Example 2. Consider again the specification of Example 1. Due to the choice operator, this specification can produce two different sequences of events, namely $\langle$`mission fail`$\rangle$ and $\langle$`mission success medal`$\rangle$. Clearly, the same sequences are produced by the Petri net generated by Algorithm 1 (shown in Fig. 3). In order to generate the Petri net, the algorithm performs two iterations; the first iteration generates the white nodes of Fig. 3 and grey nodes are generated in the second one.

Example 3 (Example revisited (computing step by step)). Now we show step by step how the execution of Algorithm 1 produced the Petri net in Fig. 3 from the CSP specification of Example 1.

This CSP specification, due to the choice operator, can produce two different sequences of events, namely $\langle$`mission fail`$\rangle$ and $\langle$`mission success medal`$\rangle$. Therefore, Algorithm 1 obtains two computations, called respectively **First iteration** and **Second iteration** in Fig. 5. In this figure, for each state, we show a sequence of rules applied from left to right to obtain the next state. We first execute the semantics with the initial state $(\texttt{MAIN}, p_0, \mathcal{N}_0, \emptyset, ([], []), \emptyset)$ and get the computation **First iteration**. This computation corresponds to the execution of the left branch of the choice with the occurrence of event `fail`. The final state is $State_8 = (\texttt{STOP}, p_1, \mathcal{N}_7, C_1, ([], S_8), \emptyset)$. Note that the stack S_8 contains a pair $(\texttt{C1}, \{\texttt{C2}\})$ to denote that the left branch of the choice has been executed. Then, the algorithm calls function `UpdStack` and executes the semantics again with the new initial state $State_9 = (\texttt{MAIN}, p_0, \mathcal{N}_7, \emptyset, (S_9, []), \emptyset)$ and it gets the computation **Second iteration**. After this execution the final Petri net ($\mathcal{N}_{10}$) has been computed. Figure 3 shows the Petri net generated where white nodes were generated in the first iteration; and grey nodes were generated in the second iteration.

For those readers interested in the complete sequence of rewriting steps performed by the semantics, we provide in Fig. 6 the complete derivations of the semantics that the algorithm fired. Each computation step is labeled with the applied rule.

5 Conclusions

This work introduces an algorithm to automatically build a Petri net which produces the same sequences of observable events that a given CSP specification. The algorithm uses an instrumentation of the standard CSP's operational semantics to explore all possible computations of a specification. The semantics is deterministic because the rule applied in every step is predetermined by the initial configuration. Therefore, the algorithm can execute the semantics several times to iteratively explore all computations and hence, generate the whole Petri net. The Petri net is generated even for non-terminating specifications due to the

First iteration

$State_0 = (\texttt{MAIN}, p_0, \mathcal{N}_0, \emptyset, ([], []), \emptyset)$ (PC)

where $\mathcal{N}_0 = (\langle \{p_0\}, \emptyset, \emptyset \rangle, M_0, \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T), M_0(p_0) = 1,$

$\mathcal{P} = Names \cup \{\Box, \texttt{STOP}\}, \mathcal{T} = \Sigma^\tau \cup \{\|, \texttt{C1}, \texttt{C2}\}$

$State_1 = (\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{\|}_{(\bullet,\emptyset,\bullet,\emptyset,\bullet)} \texttt{NASA}, p_1, \mathcal{N}_1, C_1, ([], []), \emptyset)$ (SP5)(SP1)(PC)

where $\mathcal{N}_1 = \mathcal{N}_0[p_{\texttt{MAIN}} \mapsto t_\tau \mapsto p_1], \mathcal{L}_P(p_0) = \texttt{MAIN}$ and $C_1 = \{(\texttt{MAIN}, p_1)\}$

$State_2 = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{lab_2} \texttt{NASA}, p_1, \mathcal{N}_2, C_1, ([], S_2), \emptyset)$ (SP5)(SP2)(PC)

where $\mathcal{N}_2 = \mathcal{N}_1[p_1 \mapsto t_\| \mapsto p_2, p_1 \mapsto t_\| \mapsto p_3, p_{\texttt{ASTRONAUT}} \mapsto t_\tau \mapsto p_4],$

$\mathcal{L}_P(p_2) = \texttt{ASTRONAUT}, lab_2 = (p_4, C_2, p_3, C_1, \bullet),$

$C_2 = \{(\texttt{ASTRONAUT}, p_2)\} \cup C_1$ and $S_2 = [(\texttt{SP1}, \emptyset)]$

$State_3 = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{lab_3} rhs(\texttt{NASA}), p_1, \mathcal{N}_3, C_1, ([], S_3), \emptyset)$ (SP5)(SP1)(Pref)

where $rhs(\texttt{NASA}) = (\texttt{success} \rightarrow \texttt{medal} \rightarrow \texttt{NASA}), \mathcal{N}_3 = \mathcal{N}_2[p_{\texttt{NASA}} \mapsto t_\tau \mapsto p_5],$

$\mathcal{L}_P(p_3) = \texttt{NASA}, lab_3 = (p_4, C_2, p_5, C_3, \bullet),$

$C_3 = \{(\texttt{NASA}, p_3)\} \cup C_1$ and $S_3 = (\texttt{SP2}, \emptyset) : S_2$

$State_4 = (\texttt{MISSION} \underset{\{\texttt{success}\}}{\|}_{lab_4} rhs(\texttt{NASA}), p_1, \mathcal{N}_4, C_1, ([], S_4), \Delta_1)$ (SP5)(SP1)(PC)

where $\mathcal{N}_4 = \mathcal{N}_3[p_4 \mapsto t_{\texttt{mission}} \mapsto p_6], lab_4 = (p_6, C_2, p_5, C_3, \bullet),$

$S_4 = (\texttt{SP1}, \emptyset) : S_3$ and $\Delta_1 = \{(p_4, t_{\texttt{mission}}, p_6)\}$

$State_5 = (rhs(\texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{lab_5} rhs(\texttt{NASA}), p_1, \mathcal{N}_5, C_1, ([], S_5), \emptyset)$ (SP5)(SP1)(Choice)

where $rhs(\texttt{MISSION}) = (\texttt{fail} \rightarrow \texttt{STOP}) \Box (\texttt{success} \rightarrow \texttt{STOP}),$

$\mathcal{N}_5 = \mathcal{N}_4[p_{\texttt{MISSION}} \mapsto t_\tau \mapsto p_6], \mathcal{L}_P(p_6) = \texttt{MISSION}, lab_5 = (p_7, C_4, p_5, C_3, \bullet),$

$C_4 = \{(\texttt{MISSION}, p_6)\} \cup C_2$ and $S_5 = (\texttt{SP1}, \emptyset) : S_4$

$State_6 = ((\texttt{fail} \rightarrow \texttt{STOP}) \underset{\{\texttt{success}\}}{\|}_{lab_6} rhs(\texttt{NASA}), p_1, \mathcal{N}_6, C_1, ([], S_6), \emptyset)$ (SP5)(SP1)(Pref)

where $\mathcal{N}_6 = \mathcal{N}_5[p_\Box \mapsto t_{\texttt{C1}} \mapsto p_8], \mathcal{L}_P(p_7) = \Box, lab_6 = (p_8, C_4, p_5, C_3, \bullet)$

and $S_6 = [(\texttt{C1}, \{\texttt{C2}\}), (\texttt{SP1}, \emptyset)] : S_5$

$State_7 = (\texttt{STOP} \underset{\{\texttt{success}\}}{\|}_{lab_7} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_1, ([], S_7), \Delta_2)$ (SP5)(SP4)

where $\mathcal{N}_7 = \mathcal{N}_6[p_8 \mapsto t_{\texttt{fail}} \mapsto p_9], lab_7 = (p_9, C_4, p_5, C_3, \bullet),$

$S_7 = (\texttt{SP1}, \emptyset) : S_6$ and $\Delta_2 = \{(p_8, t_{\texttt{fail}}, p_9)\}$

$State_8 = (\texttt{STOP}, p_1, \mathcal{N}_7, C_1, ([], S_8), \emptyset)$ where $S_8 = (\texttt{SP4}, \emptyset) : S_7$

Fig. 5. An example of computation with Algorithm 1

use of a loop detection mechanism controlled by the semantics. This semantics is an interesting result because it explicitly relates the CSP model with the Petri net. The way in which the semantics has been instrumented can be used for other similar purposes with slight modifications. For instance, the same design could be used to generate other graph representations of a computation [6, 7].

This is the first approach that generates the Petri net as a side-effect of an instrumented semantics. The main advantage of this new approach is that the Petri net generated is very similar (structurally) to the CSP specification.

On the practical side, we have implemented a tool called $CSP2PN$ which is able to automatically generate a Petri net equivalent to a CSP specification. This tool implements the algorithm described in this paper. However, in the implementation the algorithm is much more complex because it contains some improvements that significantly speed up the Petri net construction. The most important improvement is to avoid repeated computations. This is done by: (i) state memorization: once a state already explored is reached the algorithm stops this computation and starts with another one; and (ii) skipping already performed computations: computations do not start from `MAIN`, they start from the next non-deterministic state in the execution (this is provided by the information of the stack).

The implementation, source code and several examples are publicly available at: http://users.dsic.upv.es/ jsilva/CSP2PN/

References

1. F. de Cindio, G. de Michelis, L. Pomello and C. Simone. A Petri Net Model of CSP. In *Proc. CIL'81*, pp. 392–406, Barcelona, 1981.
2. P. Degano, R. Gorrieri and S. Marchetti. An Exercise in Concurrency: A CSP Process as a Condition/Event System. In *Advances in Petri Nets 1988*, LNCS 340, Springer-Verlag, pp. 185–105, 1988.
3. U. Goltz and W. Reisig. CSP-programs as nets with individual tokens. In *Advances in Petri Nets 1984*, LNCS 188, Springer-Verlag, pp. 169–196, 1985.
4. M. Hack. *Petri Net Languages*. Technical Report 159, Massachusetts Institute of Technology, Cambridge, MA, USA, 1976.
5. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
6. M. Leuschel, M. Llorens, J. Oliver, J. Silva, and S. Tamarit. The MEB and CEB static analysis for CSP specifications. In *Post-proceedings of the 18th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'08)*, Revised Selected Papers, LNCS 5438, Springer-Verlag, pp. 103–118, 2009.
7. M. Llorens, J. Oliver, J. Silva, and S. Tamarit. An Algorithm to Generate the Context-sensitive Synchronized Control Flow Graph. To appear in *Proc. of the 25th ACM Symposium on Applied Computing (SAC 2010)*, Sierre, Switzerland, 2010.
8. J. Magott. Performance Evaluation of Communicating Sequential Processes (CSP) using Petri Nets. In *IEE Proceedings-E*, vol. 139(3), pp. 237-241, 1992.
9. A. Mazzeo, N. Mazzocca, S. Russo, C. Savy and V. Vittorini. Formal Specification of Concurrent Systems: A Structured Approach. *The Computer Journal*, vol. 41(3), pp. 145-162, 1998.
10. R. Milner. A Calculus of Communicating Systems. LNCS 92, Springer-Verlag, 1980.
11. T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proc. of the IEEE*, vol. 77(4), pp. 541–580, IEEE Computer Society, 1989.
12. E. R. Olderog. Operational Petri Net Semantics for CCSP. In *Advances in Petri Nets 1987*, LNCS 266, Springer-Verlag, pp. 196–223, 1987.
13. J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.
14. A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 2005.

15. F. T. Sheldon. *Specification and Analysis of Stochastic Properties for Concurrent Systems Expressed Using CSP*. Ph.D. Dissertation, Computer Science and Engineering Dept, The University of Texas at Arlington, May 1996.

Second iteration

$State_9 = (\texttt{MAIN}, p_0, \mathcal{N}_7, \emptyset, (\texttt{UpdStack}(S_8), []), \emptyset) = (\texttt{MAIN}, p_0, \mathcal{N}_7, \emptyset, (S_9, []), \emptyset)$ (PC)

where $S_9 = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP2}, \emptyset), (\texttt{SP1}, \emptyset)]$

$State_{10} = (\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{\|}_{(\bullet,\emptyset,\bullet,\emptyset,\bullet)} \texttt{NASA}, p_1, \mathcal{N}_7, C_{10}, (S_9, []), \emptyset)$ (SP1)(PC)

where $\mathcal{L}_P(p_0) = \texttt{MAIN}$ and $C_{10} = \{(\texttt{MAIN}, p_1)\}$

$State_{11} = (rhs(\texttt{ASTRONAUT}) \underset{\{\texttt{success}\}}{\|}_{lab_{11}} \texttt{NASA}, p_1, \mathcal{N}_7, C_{10}, (S_{10}, [(\texttt{SP1}, \emptyset)]), \emptyset)$ (SP2)(PC)

where $rhs(\texttt{ASTRONAUT}) = (\texttt{mission} \rightarrow \texttt{MISSION})$, $\mathcal{L}_P(p_2) = \texttt{ASTRONAUT}$,

$lab_{11} = (p_4, C_{11}, p_3, C_{10}, \bullet)$, $C_{11} = \{(\texttt{ASTRONAUT}, p_2)\} \cup C_{10}$ and

$S_{10} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP2}, \emptyset)]$

$State_{12} = (rhs(\texttt{ASTRONAUT}) \underset{\{\texttt{success}\}}{\|}_{lab_{12}} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{11}, S_{12}), \emptyset)$ (SP1)(Pref)

where $rhs(\texttt{NASA}) = (\texttt{success} \rightarrow \texttt{medal} \rightarrow \texttt{NASA})$, $\mathcal{L}_P(p_3) = \texttt{NASA}$,

$lab_{12} = (p_4, C_{11}, p_5, C_{12}, \bullet)$, $C_{12} = \{(\texttt{NASA}, p_3)\} \cup C_{10}$,

$S_{11} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset)]$ and $S_{12} = [(\texttt{SP2}, \emptyset) : (\texttt{SP1}, \emptyset)]$

$State_{13} = (\texttt{MISSION} \underset{\{\texttt{succes}\}}{\|}_{lab_{13}} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{13}, S_{14}), \Delta_1)$ (SP1)(PC)

where $lab_{13} = (p_6, C_{11}, p_5, C_{12}, \bullet)$, $S_{13} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset)]$,

$S_{14} = (\texttt{SP1}, \emptyset) : S_{12}$ and $\Delta_1 = \{(p_4, t_{\texttt{mission}}, p_6)\}$

$State_{14} = (rhs(\texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{lab_{14}} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{15}, S_{16}), \emptyset)$ (SP1)(Choice)

where $rhs(\texttt{MISSION}) = ((\texttt{fail} \rightarrow \texttt{STOP}) \Box (\texttt{success} \rightarrow \texttt{STOP}))$, $\mathcal{L}_P(p_6) = \texttt{MISSION}$,

$lab_{14} = (p_7, C_{13}, p_5, C_{12}, \bullet)$, $C_{13} = \{(\texttt{MISSION}, p_6)\} \cup C_{11}$,

$S_{15} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset)]$ and $S_{16} = (\texttt{SP1}, \emptyset) : S_{14}$

$State_{15} = ((\texttt{success} \rightarrow \texttt{STOP}) \underset{\{\texttt{success}\}}{\|}_{lab_{15}} rhs(\texttt{NASA}), p_1, \mathcal{N}_8, C_{10}, ([], S_{17}), \emptyset)$ (SP5)(SP3)(Pref)(Pref)

where $\mathcal{N}_8 = \mathcal{N}_7[p_\Box \mapsto t_{\texttt{C2}} \mapsto p_{10}]$, $\mathcal{L}_P(p_7) = \Box$,

$lab_{15} = (p_{10}, C_{13}, p_5, C_{12}, \bullet)$ and $S_{17} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset)] : S_{16}$

$State_{16} = (\texttt{STOP} \underset{\{\texttt{success}\}}{\|}_{lab_{16}} (\texttt{medal} \rightarrow \texttt{STOP}), p_1, \mathcal{N}_9, C_{10}, ([], S_{18}), \Delta)$ (SP5)(SP2)(Pref)

where $\mathcal{N}_9 = \mathcal{N}_8[p_{10} \mapsto t_{\texttt{success}} \mapsto p_{11}, p_5 \mapsto t_{\texttt{success}} \mapsto p_{12}]$,

$lab_{16} = (p_{11}, C_{13}, p_{12}, C_{12}, \bullet)$, $S_{18} = (\texttt{SP3}, \emptyset) : S_{17}$ and

$\Delta = \{(p_{10}, t_{\texttt{success}}, p_{11})\} \cup \{(p_5, t_{\texttt{success}}, p_{12})\}$

$State_{17} = (\texttt{STOP} \underset{\{\texttt{success}\}}{\|}_{lab_{17}} \texttt{STOP}, p_1, \mathcal{N}_{10}, C_{10}, ([], S_{19}), \Delta')$ (SP5)(SP4)

where $lab_{17} = (p_{11}, C_{13}, p_{13}, C_{12}, \bullet)$, $S_{19} = (\texttt{SP2}, \emptyset) : S_{18}$ and

$\Delta' = \{p_{11}, t_{\texttt{medal}}, p_{13}\}$

$State_{18} = (\texttt{STOP}, p_1, \mathcal{N}_{10}, C_{10}, ([], S_{20}), \emptyset)$

where $S_{20} = (\texttt{SP4}, \emptyset) : S_{19} =$

$[(\texttt{SP4}, \emptyset), (\texttt{SP2}, \emptyset), (\texttt{SP3}, \emptyset), (\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP2}, \emptyset), (\texttt{SP1}, \emptyset)]$

$State_{19} = (\texttt{MAIN}, p_0, \mathcal{N}_{10}, \emptyset, (\texttt{UpdStack}(S_{20}), []), \emptyset) = (\texttt{MAIN}, p_0, \mathcal{N}_{10}, \emptyset, ([], []), \emptyset)$

Fig. 5. An example of computation with Algorithm 1 (cont.)

$$\text{(PC)}\ \frac{}{(\texttt{MAIN}, p_0, \mathcal{N}_0, \emptyset, ([\,],[\,]), \emptyset) \xrightarrow{\tau} State_1} \text{ where}$$

$$\mathcal{N}_0 = (\langle\{p_0\}, \emptyset, \emptyset\rangle, M_0, \mathcal{P}, \mathcal{T}, \mathcal{L}_P, \mathcal{L}_T), M_0(p_0) = 1, \mathcal{P} = Names \cup \{\square, \texttt{STOP}\}, \mathcal{T} = \Sigma^\tau \cup \{\|, \texttt{C1}, \texttt{C2}\},$$

$$State_1 = (\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{\|}_{(\bullet,\emptyset,\bullet,\emptyset,\bullet)} \texttt{NASA}, p_1, \mathcal{N}_1, C_1, ([\,],[\,]), \emptyset), \mathcal{N}_1 = \mathcal{N}_0[p_{\texttt{MAIN}} \mapsto t_\tau \mapsto p_1], \mathcal{L}_P(p_0) = \texttt{MAIN} \text{ and } C_1 = \{(\texttt{MAIN}, p_1)\}$$

$$\text{(SP5)}\ \frac{\text{(SP1)}\ \dfrac{\text{(PC)}\ \dfrac{}{(\texttt{ASTRONAUT}, p_2, \mathcal{N}_1[p_1 \mapsto t_\| \mapsto p_2, p_1 \mapsto t_\| \mapsto p_3], C_1, ([\,], [(\texttt{SP1}, \emptyset)]), \emptyset) \xrightarrow{\tau} (\texttt{mission} \rightarrow \texttt{MISSION}, p_4, \mathcal{N}_1[p_{\texttt{ASTRONAUT}} \mapsto t_\tau \mapsto p_4], C_2, ([\,], S_2), \emptyset)}}{(\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{\|}_{(\bullet,\emptyset,\bullet,\emptyset,\bullet)} \texttt{NASA}, p_1, \mathcal{N}_1, C_1, ([(\texttt{SP1}, \emptyset)], [\,]), \emptyset) \xrightarrow{\tau} State_2}}{State_1 \xrightarrow{\tau} State_2}$$

$$\text{where } State_2 = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{(p_4, C_2, p_3, C_1, \bullet)} \texttt{NASA}, p_1, \mathcal{N}_2, C_1, ([\,], S_2), \emptyset),$$

$$\mathcal{L}_P(p_2) = \texttt{ASTRONAUT}, C_2 = \{(\texttt{ASTRONAUT}, p_2)\} \cup C_1 \text{ and } S_2 = [(\texttt{SP1}, \emptyset)]$$

$$\text{(SP5)}\ \frac{\text{(SP2)}\ \dfrac{\text{(PC)}\ \dfrac{}{(\texttt{NASA}, p_3, \mathcal{N}_2, C_1, ([\,], (\texttt{SP2}, \emptyset) : S_2), \emptyset) \xrightarrow{\tau} (rhs(\texttt{NASA}), p_5, \mathcal{N}_2[p_{\texttt{NASA}} \mapsto t_\tau \mapsto p_5], C_3, ([\,], S_3), \emptyset)}}{(\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{(p_4, C_2, p_3, C_1, \bullet)} \texttt{NASA}, p_1, \mathcal{N}_2, C_1, ([(\texttt{SP2}, \emptyset)], S_2), \emptyset) \xrightarrow{\tau} State_3}}{State_2 \xrightarrow{\tau} State_3}$$

$$\text{where } State_3 = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{(p_4, C_2, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_3, C_1, ([\,], S_3), \emptyset),$$

$$rhs(\texttt{NASA}) = (\texttt{success} \rightarrow \texttt{medal} \rightarrow \texttt{NASA}), \mathcal{L}_P(p_3) = \texttt{NASA}, C_3 = \{(\texttt{NASA}, p_3)\} \cup C_1 \text{ and } S_3 = (\texttt{SP2}, \emptyset) : S_2$$

$$\text{(SP5)}\ \frac{\text{(SP1)}\ \dfrac{\text{(Pref)}\ \dfrac{}{((\texttt{mission} \rightarrow \texttt{MISSION}), p_4, \mathcal{N}_3, C_2, ([\,], [(\texttt{SP1}, \emptyset)] : S_3), \emptyset) \xrightarrow{\texttt{mission}} (\texttt{MISSION}, p_6, \mathcal{N}_3[p_4 \mapsto t_{\texttt{mission}} \mapsto p_6], C_2, ([\,], S_4), \Delta_1)}}{((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\|}_{(p_4, C_2, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_3, C_1, ([(\texttt{SP1}, \emptyset)], S_3), \emptyset) \xrightarrow{\texttt{mission}} State_4}}{State_3 \xrightarrow{\texttt{mission}} State_4}$$

$$\text{where } State_4 = (\texttt{MISSION} \underset{\{\texttt{succes}\}}{\|}_{(p_6, C_2, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_4, C_1, ([\,], S_4), \Delta_1), S_4 = (\texttt{SP1}, \emptyset) : S_3 \text{ and } \Delta_1 = \{(p_4, t_{\texttt{mission}}, p_6)\}$$

Fig. 6. A computation with the instrumented semantics in Fig. 4

$$
\text{(SP5)}\,\dfrac{\text{(SP1)}\,\dfrac{\text{(PC)}\,\dfrac{}{(\texttt{MISSION}, p_6, \mathcal{N}_4, C_2, ([], [(\texttt{SP1}, \emptyset)] : S_4), \Delta_1) \xrightarrow{\tau} ((\texttt{fail} \to \texttt{STOP})\Box(\texttt{success} \to \texttt{STOP}), p_7, \mathcal{N}_4[p_{\texttt{MISSION}} \mapsto t_\tau \mapsto p_7], C_4, ([], S_5), \emptyset)}}{(\texttt{MISSION} \underset{\{\texttt{success}\}}{\|}{}_{(p_6, C_2, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_4, C_1, ([(\texttt{SP1}, \emptyset)], S_4), \Delta_1) \xrightarrow{\tau} State_5}}{State_4 \xrightarrow{\tau} State_5}
$$

where $State_5 = (((\texttt{fail} \to \texttt{STOP})\Box(\texttt{success} \to \texttt{STOP})) \underset{\{\texttt{success}\}}{\|}{}_{(p_7, C_4, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_5, C_1, ([], S_5), \emptyset)$,

$\mathcal{L}_P(p_6) = \texttt{MISSION}, C_4 = \{(\texttt{MISSION}, p_6)\} \cup C_2$ and $S_5 = (\texttt{SP1}, \emptyset) : S_4$

$$
\text{(SP5)}\,\dfrac{\text{(SP1)}\,\dfrac{\text{(Choice)}\,\dfrac{}{((\texttt{fail} \to \texttt{STOP}) \Box (\texttt{success} \to \texttt{STOP}), p_7, \mathcal{N}_5, C_4, ([], (\texttt{SP1}, \emptyset) : S_5), \emptyset) \xrightarrow{\tau} (\texttt{fail} \to \texttt{STOP}, p_8, \mathcal{N}_5[p_\Box \mapsto t_{\texttt{C1}} \mapsto p_8], C_4, ([], S_6), \emptyset)}}{((\texttt{fail} \to \texttt{STOP})\Box(\texttt{success} \to \texttt{STOP}) \underset{\{\texttt{success}\}}{\|}{}_{(p_7, C_4, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_5, C_1, ([(\texttt{SP1}, \emptyset)], S_5), \emptyset) \xrightarrow{\tau} State_6}}{State_5 \xrightarrow{\tau} State_6}
$$

where $State_6 = ((\texttt{fail} \to \texttt{STOP}) \underset{\{\texttt{success}\}}{\|}{}_{(p_8, C_4, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_6, C_1, ([], S_6), \emptyset)$,

$\mathcal{L}_P(p_7) = \Box$ and $S_6 = [(\texttt{C1}, \{\texttt{C2}\}), (\texttt{SP1}, \emptyset)] : S_5$

$$
\text{(SP5)}\,\dfrac{\text{(SP1)}\,\dfrac{\text{(Pref)}\,\dfrac{}{((\texttt{fail} \to \texttt{STOP}), p_8, \mathcal{N}_6, C_5, ([], (\texttt{SP1}, \emptyset) : S_6), \emptyset) \xrightarrow{\texttt{fail}} (\texttt{STOP}, p_9, \mathcal{N}_6[p_8 \mapsto t_{\texttt{fail}} \mapsto p_9], C_5, ([], S_7), \Delta_2)}}{((\texttt{fail} \to \texttt{STOP}) \underset{\{\texttt{success}\}}{\|}{}_{(p_8, C_5, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_6, C_1, ([(\texttt{SP1}, \emptyset)], S_6), \emptyset) \xrightarrow{\texttt{fail}} State_7}}{State_6 \xrightarrow{\texttt{fail}} State_7}
$$

where $State_7 = (\texttt{STOP} \underset{\{\texttt{success}\}}{\|}{}_{(p_9, C_4, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_1, ([], S_7), \Delta_2)$,

$S_7 = (\texttt{SP1}, \emptyset) : S_6$ and $\Delta_2 = \{(p_8, t_{\texttt{fail}}, p_9)\}$

$$
\text{(SP5)}\,\dfrac{\text{(SP4)}\,\dfrac{}{(\texttt{STOP} \underset{\{\texttt{success}\}}{\|}{}_{(p_9, C_4, p_5, C_3, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_1, ([(\texttt{SP4}, \emptyset)], S_7), \Delta_2) \xrightarrow{\tau} State_8}}{State_7 \xrightarrow{\tau} State_8}
$$

where $State_8 = (\texttt{STOP}, p_1, \mathcal{N}_7, C_1, ([], S_8), \emptyset)$ and $S_8 = (\texttt{SP4}, \emptyset) : S_7$

Fig. 6. A computation with the instrumented semantics in Fig. 4 (cont.)

$$State_9 = (\texttt{MAIN}, p_0, \mathcal{N}_7, \emptyset, (\texttt{UpdStack}(S_8), []), \emptyset) = (\texttt{MAIN}, p_0, \mathcal{N}_7, \emptyset, (S_9, []), \emptyset) \text{ where } S_9 = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP2}, \emptyset), (\texttt{SP1}, \emptyset)]$$

$$\textsf{(PC)}\frac{}{State_9 \xrightarrow{\tau} State_{10}} \text{ where}$$

$$State_{10} = (\texttt{ASTRONAUT} \underset{\{\texttt{success}\}}{\parallel}{}_{(\bullet,\emptyset,\bullet,\emptyset,\bullet)} \texttt{NASA}, p_1, \mathcal{N}_7[p_{\texttt{MAIN}} \mapsto t_\tau \mapsto p_1], C_{10}, (S_9, []), \emptyset), \mathcal{L}_P(p_0) = \texttt{MAIN} \text{ and } C_{10} = \{(\texttt{MAIN}, p_1)\}$$

$$\textsf{(SP1)}\frac{\textsf{(PC)}\dfrac{}{(\texttt{ASTRONAUT}, p_2, \mathcal{N}_7[p_1 \mapsto t_\parallel \mapsto p_2, p_1 \mapsto t_\parallel \mapsto p_3], C_{10}, (S_{10}, [(\texttt{SP1}, \emptyset)]), \emptyset) \xrightarrow{\tau} (\texttt{mission} \rightarrow \texttt{MISSION}, p_4, \mathcal{N}_7[p_{\texttt{ASTRONAUT}} \mapsto t_\tau \mapsto p_4], C_{11}, (S_{10}, [(\texttt{SP1}, \emptyset)]), \emptyset)}}{State_{10} \xrightarrow{\tau} State_{11}}$$

$$\text{where } State_{11} = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\parallel}{}_{(p_4, C_{11}, p_3, C_{10}, \bullet)} \texttt{NASA}, p_1, \mathcal{N}_7, C_{10}, (S_{10}, [(\texttt{SP1}, \emptyset)]), \emptyset),$$

$$\mathcal{L}_P(p_2) = \texttt{ASTRONAUT}, C_{11} = \{(\texttt{ASTRONAUT}, p_2)\} \cup C_{10} \text{ and } S_{10} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP2}, \emptyset)]$$

$$\textsf{(SP2)}\frac{\textsf{(PC)}\dfrac{}{(\texttt{NASA}, p_3, \mathcal{N}_7, C_{10}, (S_{11}, S_{12}), \emptyset) \xrightarrow{\tau} (rhs(\texttt{NASA}), p_5, \mathcal{N}_7[p_{\texttt{NASA}} \mapsto t_\tau \mapsto p_5], C_{12}, (S_{11}, S_{12}), \emptyset)}}{State_{11} \xrightarrow{\tau} State_{12}}$$

$$\text{where } State_{12} = ((\texttt{mission} \rightarrow \texttt{MISSION}) \underset{\{\texttt{success}\}}{\parallel}{}_{(p_4, C_{11}, p_5, C_{12}, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{11}, S_{12}), \emptyset), rhs(\texttt{NASA}) = (\texttt{success} \rightarrow \texttt{medal} \rightarrow \texttt{NASA}),$$

$$\mathcal{L}_P(p_3) = \texttt{NASA}, C_{12} = \{(\texttt{NASA}, p_3)\} \cup C_{10}, S_{11} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset)] \text{ and } S_{12} = [(\texttt{SP2}, \emptyset) : (\texttt{SP1}, \emptyset)]$$

$$\textsf{(SP1)}\frac{\textsf{(Pref)}\dfrac{}{((\texttt{mission} \rightarrow \texttt{MISSION}), p_4, \mathcal{N}_7, C_{11}, (S_{13}, S_{14}), \emptyset) \xrightarrow{\texttt{mission}} (\texttt{MISSION}, p_6, \mathcal{N}_7[p_4 \mapsto t_{\texttt{mission}} \mapsto p_6], C_{11}, (S_{13}, S_{14}), \Delta_1)}}{State_{12} \xrightarrow{\texttt{mission}} State_{13}}$$

$$\text{where } State_{13} = (\texttt{MISSION} \underset{\{\texttt{succes}\}}{\parallel}{}_{(p_6, C_{11}, p_5, C_{12}, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{13}, S_{14}), \Delta_1),$$

$$S_{13} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset), (\texttt{SP1}, \emptyset)], S_{14} = (\texttt{SP1}, \emptyset) : S_{12} \text{ and } \Delta_1 = \{(p_4, t_{\texttt{mission}}, p_6)\}$$

$$\textsf{(SP1)}\frac{\textsf{(PC)}\dfrac{}{(\texttt{MISSION}, p_6, \mathcal{N}_7, C_{11}, (S_{15}, [(\texttt{SP1}, \emptyset)] : S_{14}), \Delta_1) \xrightarrow{\tau} ((\texttt{fail} \rightarrow \texttt{STOP}) \Box (\texttt{success} \rightarrow \texttt{STOP}), p_7, \mathcal{N}_7[p_{\texttt{MISSION}} \mapsto t_\tau \mapsto p_7], C_{13}, (S_{15}, S_{16}), \emptyset)}}{State_{13} \xrightarrow{\tau} State_{14}}$$

$$\text{where } State_{14} = (((\texttt{fail} \rightarrow \texttt{STOP}) \Box (\texttt{success} \rightarrow \texttt{STOP})) \underset{\{\texttt{success}\}}{\parallel}{}_{(p_7, C_{13}, p_5, C_{12}, \bullet)} rhs(\texttt{NASA}), p_1, \mathcal{N}_7, C_{10}, (S_{15}, S_{16}), \emptyset),$$

$$\mathcal{L}_P(p_6) = \texttt{MISSION}, C_{13} = \{(\texttt{MISSION}, p_6)\} \cup C_{11}, S_{15} = [(\texttt{C2}, \emptyset), (\texttt{SP1}, \emptyset)] \text{ and } S_{16} = (\texttt{SP1}, \emptyset) : S_{14}$$

Fig. 6. A computation with the instrumented semantics in Fig. 4 (cont.)

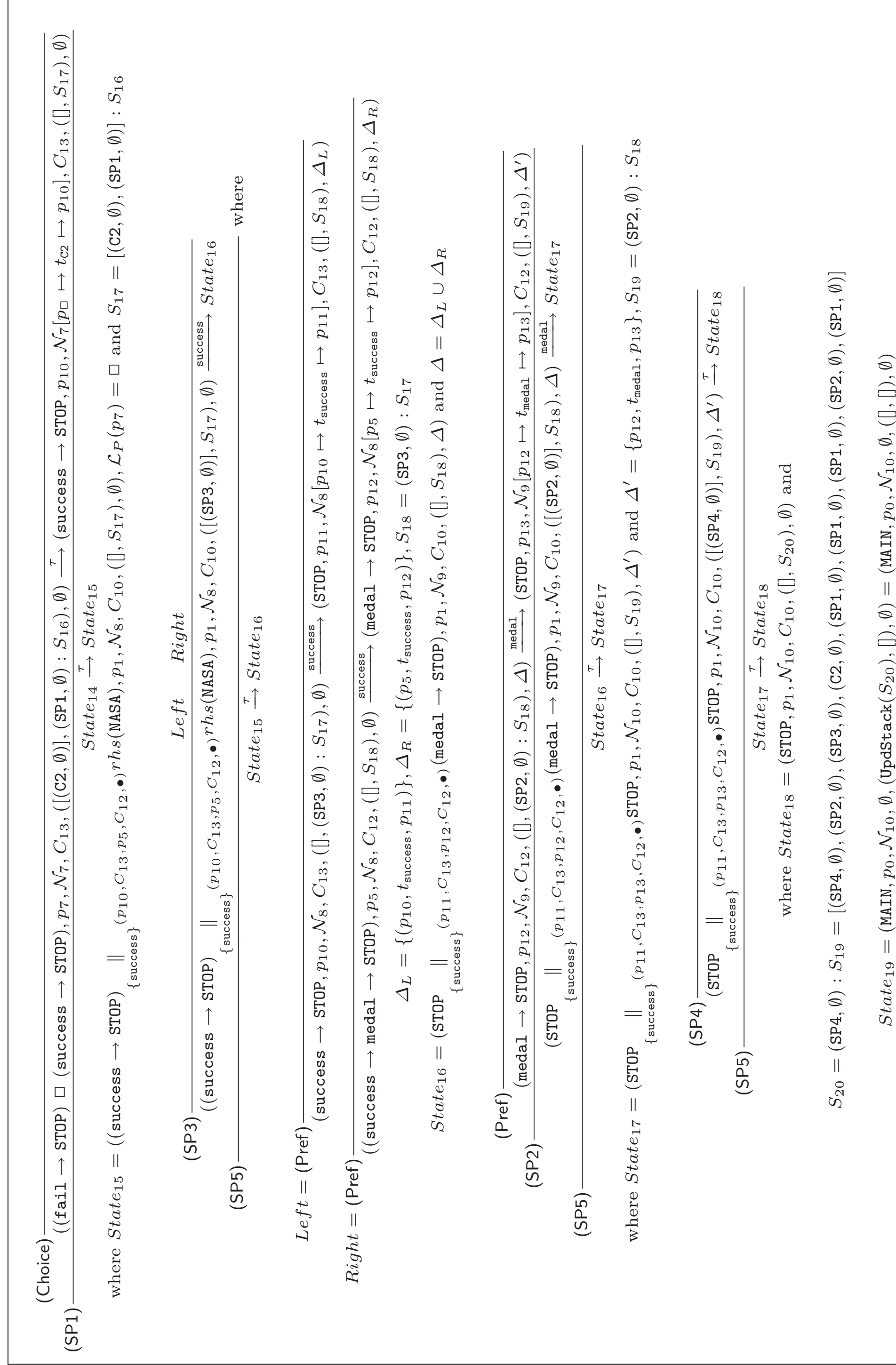

Fig. 6. A computation with the operational semantics in Fig. 4 (cont.)