On the Complexity of Some Verification Problems in Process Control Specifications

A.H.M. ter Hofstede[1] and M.E. Orlowska[2]

[1]Cooperative Information Systems Research Centre
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434
Brisbane Qld 4001
Australia
e-mail: arthur@icis.qut.edu.au

[2]School of Information Technology
The University of Queensland
Brisbane, Qld 4072
Australia
e-mail: maria@cs.uq.oz.au

Abstract

Many techniques in many diverse areas in computer science, such as process modelling, process programming, decision support systems, workflow systems, use concepts for the specification of process control. These typically include concepts for the specification of sequential execution, parallelism, synchronisation, and moments of choice. This paper identifies some typical verification problems in process control specifications, such as reachability, termination, freedom of deadlock and livelock, and determines their computational complexity. These results then provide computational lower bounds for any technique using these, but possibly more, concepts for process control.

Keywords: Control Flow, Verification, Computational Complexity

1 Introduction

In many specification techniques constructs for modelling process control occur. Examples of such techniques can be found in such diverse fields as Decision Support Systems (e.g. [2, 7]), Process Programming (e.g. [18]), Process Modelling (e.g. [6, 9]), life-cycle models in Object-Orientation (e.g. [14, 12]), Meta Process Modelling (e.g. [21, 15]) and the modelling of workflows (e.g. [4]). Although many of the techniques involved offer quite domain specific concepts, each of them contains a number of core concepts for the representation of process control flow.

These core concepts typically allow for the specification of sequential execution, parallelism, synchronisation, iteration, choice and process decomposition. The focus of this paper is on the identification and the computational complexity of some essential verification problems in process control specifications. These verification problems include reachability, termination, and freedom of deadlock and livelock. The complexity results then provide computational lower bounds for any technique using these aforementioned concepts.

2 Definition of Essential Process Control Concepts

In essence, process control specifications deal with the representation of execution dependencies between processes. A process control specification language should at least be capable of capturing moments of choice, sequential composition, parallel execution, and synchronization. In this section the core concepts for specifying these execution dependencies are identified and formally defined.

2.1 Identification of Process Control Concepts

The most elementary notion in a process control specification is that of a *process*. Informally, a process is something that is to be performed in order to achieve a certain goal. Processes typically have names. Processes with identical names represent processes with identical functionality. Some names may represent atomic actions, while other names may have an associated decomposition. In general, recursive decomposition is allowed, as this facilitates the specification of recursive processes. Due to iterative structures, it has to be specified explicitly which process control objects are to be performed first as part of the execution of a decomposed process. Such process control objects are referred to as *initial items*.

Triggers represent sequential order, if a process control object A has a trigger to a process control object B, termination of A implies start of the execution of B. A process control object may have multiple output triggers, in which case all these process control objects have to be performed in parallel. *Decisions* represent moments of choice, they coordinate the execution of tasks. Typically, a distinction is made between decisions with a termination option and decisions without that option. A choice for termination implies termination of that particular execution path. If that execution path is the only active execution path, the process as a whole terminates as well. *Synchronizers* represent synchronisation points. A synchronizer is activated if and only if all its input process control objects have terminated.

Figure 1 contains an overview of graphical representations of the core process control objects in order to facilitate the presentation of examples.

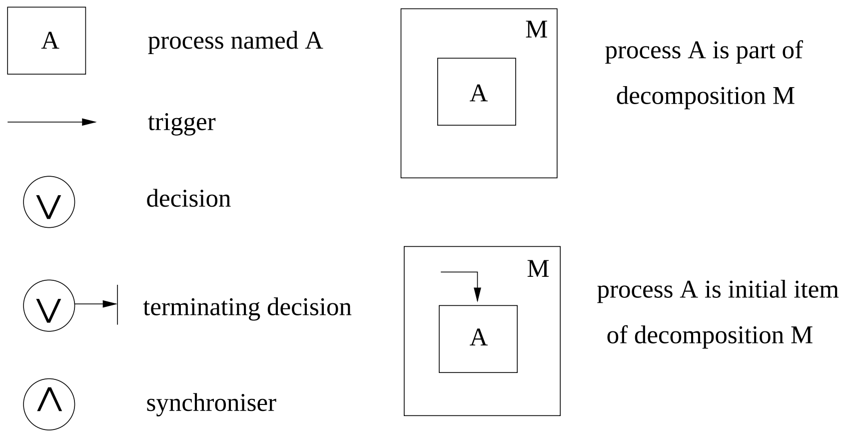

Figure 1: Legend of graphical representations of process control concepts

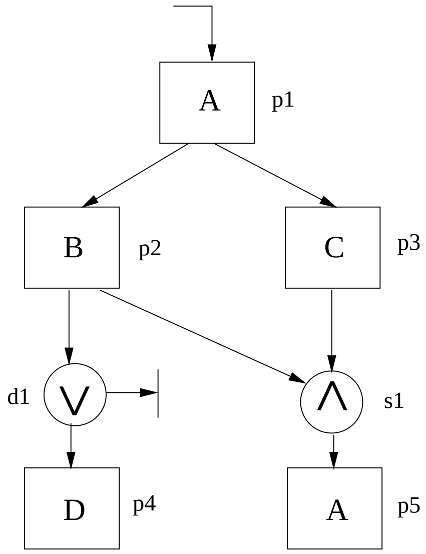

Figure 2: Example of a process control specification

Example 2.1 As a simple example of a process control specification consider figure 2. This process control specifcation, which does not use decomposition, has process p_1 with name A as initial item. Hence, it starts with the execution of this process. After the execution of p_1, processes p_2 and p_3 run in parallel. After termination of p_2, a decision is taken whether to perform p_4 or not. After both p_2 and p_3 have terminated, process p_5 is executed.

□

2.2 Syntax of Process Control Specifications

Formally, a process control specification $\mathcal{W}$ consists of the following components:

1. A set $\mathcal{X}$ of process control objects. $\mathcal{X}$ is the union of a set of synchronisers $\mathcal{S}$, a set of processes $\mathcal{P}$ and a set of decisions $\mathcal{D}$. In $\mathcal{D}$ we distinguish a subset $\mathcal{D}_t$ consisting of the terminating decisions. The auxiliary set of non-synchronisers $\mathcal{U}$ is defined as $\mathcal{P} \cup \mathcal{D}$.

2. A relation $\mathsf{Trig} \subseteq \mathcal{X} \times \mathcal{X}$ of triggers.

3. A function $\mathsf{Name}\colon \mathcal{P} \to \mathcal{V}$ yielding the name of a process, where $\mathcal{V}$ is a set of names.

4. A partial decomposition function $\mathsf{Sup}\colon \mathcal{X} \rightarrowtail \mathcal{V}$. If $\mathsf{Sup}(x) = v$, this means that process control object x is part of the decomposition of v. Names in the range of this function are called *decomposition names*. The set $\mathcal{Q} \equiv \mathsf{ran}(\mathsf{Sup})$ contains all these names. The complement $\mathcal{A} \equiv \mathcal{V} \setminus \mathcal{Q}$ is the set of *atomic actions*.

5. A set $\mathcal{I} \subseteq \mathcal{X}$ containing the initial items.

The process control objects that have to be executed at the start of a process control specification $\mathcal{W}$ are those process control objects that are initial item, but do not have a supertask.

The set containing these objects is referred to as $\mathcal{J}$ and is defined as:

$$\{x \in \mathcal{I} \mid \mathsf{Sup}(x)\uparrow\}$$

Decomposition allows for the modularization of process control specifications. Hence, triggers should not cross decomposition boundaries:

$$\mathsf{Trig}(x_1, x_2) \Rightarrow \mathsf{Sup}(x_1) = \mathsf{Sup}(x_2)$$

Finally, the *size* of a process control specification $\mathcal{W}$ is defined to be $|\mathcal{X}| + |\mathsf{Trig}| + |\mathsf{Sup}| + |\mathcal{I}|$.

2.3 Semantics of Process Control Specifications

In this section the semantics of process control specifications is defined by means of a translation to Process Algebra [1]. This translation is based on the translation of Task Structures in [11]. As an algebraic theory Process Algebra belongs to the same family as CCS [17] and CSP [10].

2.3.1 Introduction to ACP$_\varepsilon$

Although the name Process Algebra suggests a single algebra to describe processes, it actually refers to a whole family of algebras based on the same principles. Traditionally, only the family member used is presented. In this section we briefly introduce the Algebra of Communicating Processes with the empty action (ACP$_\varepsilon$) as the semantics of process control specifications will be defined in this algebra. For an in-depth treatment we refer to [1]. The axioms of ACP$_\varepsilon$ are listed in table 1.

The units of Process Algebra are atomic actions. The set of all atomic actions is called $\mathcal{A}$. Although they are units of calculation, atomic actions need not be indivisible (see [8]). Starting with atomic actions, new processes can be constructed by applying sequential and alternative composition ("$\cdot$" resp. "$+$"). The algebra that results (axioms A1-A5) is called basic process algebra (BPA). As a convention, the names of atomic actions are written in lowercase (e.g. a, b, red_nose_reindeer), while process variables are written in uppercase (e.g. A, B, RUDOLPH). Normally, the $\cdot$ will be omitted unless this results in ambiguity. A special constant δ, *deadlock*, denotes the inaction, or impossibility to proceed (axioms A6-A7).

To add parallelism, an additional operator has to be introduced. This operator, called (free) *merge* and denoted as $\|$, is defined with the aid of an auxiliary operator $\mathbin{\|\!\_}$, the *left-merge* (axioms C1-C4).

Another special constant ε, the *empty action*, is used to denote the process that does nothing but terminate successfully (axioms A8-A9). After adding ε, processes may terminate directly. The *termination operator* $\surd$ determines whether or not this termination option is present for a given process (axioms TE1-TE4). The inclusion of the expression $\surd(X)\,\surd(Y)$ in axiom C1 guarantees that $\varepsilon \,\|\, \varepsilon = \varepsilon$.

The communication merge $|$ allows parallel processes to exchange information, i.e. to communicate (axioms C5-C9). Associated with communication is a *communication function* γ defined over pairs of atomic actions (axioms C10-C11). Specific process specifications will have to define

$X+Y=Y+X$	A1	$X+\delta=X$	A6
$(X+Y)+Z=X+(Y+Z)$	A2	$\delta\cdot X=\delta$	A7
$X+X=X$	A3	$X\cdot\varepsilon=X$	A8
$(X+Y)\cdot Z=X\cdot Z+Y\cdot Z$	A4	$\varepsilon\cdot X=X$	A9
$(X\cdot Y)\cdot Z=X\cdot(Y\cdot Z)$	A5		
$X \parallel Y = X \mathbin{\lfloor\!\lfloor} Y + Y \mathbin{\lfloor\!\lfloor} X + X \mid Y + \surd(X)\,\surd(Y)$	C1	$aX \mid bY = (a \mid b)(X \parallel Y)$	C7
$\varepsilon \mathbin{\lfloor\!\lfloor} X = \delta$	C2	$(X+Y) \mid Z = X \mid Z + Y \mid Z$	C8
$aX \mathbin{\lfloor\!\lfloor} Y = a(X \parallel Y)$	C3	$X \mid (Y+Z) = X \mid Y + X \mid Z$	C9
$(X+Y) \mathbin{\lfloor\!\lfloor} Z = X \mathbin{\lfloor\!\lfloor} Z + Y \mathbin{\lfloor\!\lfloor} Z$	C4	$a \mid b = \gamma(a,b)$ if $\gamma(a,b)$ defined	C10
$\varepsilon \mid X = \delta$	C5	$a \mid b = \delta$ otherwise	C11
$X \mid \varepsilon = \delta$	C6		
$\surd(\varepsilon)=\varepsilon$	TE1	$\partial_H(a)=a$ if $a\notin H$	D1
$\surd(a)=\delta$	TE2	$\partial_H(a)=\delta$ if $a\in H$	D2
$\surd(X+Y)=\surd(X)+\surd(Y)$	TE3	$\partial_H(X+Y)=\partial_H(X)+\partial_H(Y)$	D3
$\surd(X\cdot Y)=\surd(X)\cdot\surd(Y)$	TE4	$\partial_H(X\cdot Y)=\partial_H(X)\cdot\partial_H(Y)$	D4

Table 1: Algebra of Communicating Processes with the empty action

the (partial) communication function γ. This function is both commutative and associative. Axioms C10 and C11 assume that communication is binary, but higher order communication is permitted as well.

Finally, one needs a way to prevent the isolated occurrence of atomic actions meant to communicate with other actions. This is achieved through the *encapsulation operator* ∂_H. In fact it is a whole family of operators, one for each $H \subseteq \mathcal{A}$ (axioms D1-D4).

In table 2, the action relations for ACP_ε are given. Action relations provide an operational semantics for ACP_ε terms and specify which actions a certain process can perform. An action relation $t \xrightarrow{a} s$ specifies that a process t can execute an action a thereby turning into s. The action relation $t \xrightarrow{a} \surd$ denotes that t can terminate by executing a.

The generalized action relations are given in table 3. Any sequence σ such that either $t \overset{\sigma}{\twoheadrightarrow} s$ for some s or $t \overset{\sigma}{\twoheadrightarrow} \surd$ is called a *trace* of t. In the rest of this paper $t \twoheadrightarrow s$ is used as an abbreviation for $\exists_{\sigma\in\mathcal{A}^*}\left[t \overset{\sigma}{\twoheadrightarrow} s\right]$. Likewise, $t \not\twoheadrightarrow s$ is an abbreviation for $\neg\exists_{\sigma\in\mathcal{A}^*}\left[t \overset{\sigma}{\twoheadrightarrow} s\right]$.

2.3.2 Process Algebra Semantics of Process Control Specifications

In this section the formal semantics of process control specifications is defined by means of a mapping to expressions in ACP_ε. For every process control object $x \in \mathcal{X}$, we introduce an object E_x. Intuitively, E_x should be seen as an entry point for process control object x.

$a \xrightarrow{a} \surd$		
$x \xrightarrow{a} x'$	$\Rightarrow$	$x + y \xrightarrow{a} x'$ and $y + x \xrightarrow{a} x'$
$x \xrightarrow{a} x'$	$\Rightarrow$	$xy \xrightarrow{a} x'y$
$x{\downarrow}$ and $y \xrightarrow{a} y'$	$\Rightarrow$	$xy \xrightarrow{a} y'$
$\varepsilon{\downarrow}$		
$x{\downarrow}$	$\Rightarrow$	$(x+y){\downarrow}$ and $(y+x){\downarrow}$
$x{\downarrow}$ and $y{\downarrow}$	$\Rightarrow$	$(xy){\downarrow}$
$x \xrightarrow{a} x'$	$\Rightarrow$	$x \parallel y \xrightarrow{a} x' \parallel y, y \parallel x \xrightarrow{a} y \parallel x'$ and $x \mathbin{\lfloor\!\lfloor} y \xrightarrow{a} x' \parallel y$
$x{\downarrow}$ and $y{\downarrow}$	$\Rightarrow$	$(x \parallel y){\downarrow}$
$x{\downarrow}$	$\Rightarrow$	$\surd(x){\downarrow}$
$x \xrightarrow{a} x', y \xrightarrow{b} y'$ and $\gamma(a,b) = c$	$\Rightarrow$	$x \parallel y \xrightarrow{c} x' \parallel y'$

Table 2: Action relations for the Algebra of Communicating Processes with the empty action

$t \xrightarrow{a} s$	$\Rightarrow$	$t \overset{a}{\twoheadrightarrow} s$
$t \overset{\sigma}{\twoheadrightarrow} s$ and $s \overset{\rho}{\twoheadrightarrow} r$	$\Rightarrow$	$t \overset{\sigma\rho}{\twoheadrightarrow} r$
$t \xrightarrow{a} \surd$	$\Rightarrow$	$t \overset{a}{\twoheadrightarrow} \surd$
$t \overset{\sigma}{\twoheadrightarrow} s$ and $s \overset{\rho}{\twoheadrightarrow} \surd$	$\Rightarrow$	$t \overset{\sigma\rho}{\twoheadrightarrow} \surd$

Table 3: Generalized action relations

For every process $p \in \mathcal{P}$ the corresponding process algebra equation expresses that the associated action, reflected by the name of that process, which either is atomic or corresponds to a decomposition, has to be performed and that afterwards all process control objects to be triggered are triggered in parallel. The treatment of synchronizers brings a slight complication as in order to become active they may have to await termination of other process control objects. This problem is solved through a rather unusual use of the communication operator. Every process control object x which triggers synchroniser w starts an atomic action $\alpha_{x,w}$ after it is ready. A synchroniser is started upon availability of all the atomic actions of its input process control objects. Therefore, the communication function has to be defined for the α's of each synchroniser. As a synchroniser can have more than two input process control objects, communication is not necessarily binary, which is quite uncommon indeed. Binary communication could be used at the expense of some additional definitions. The equation for a process $p \in \mathcal{P}$ is:

$$E_p = \mathsf{Name}(p) \cdot \left(\mathop{\Big\|}_{u \in \mathcal{U}, \mathsf{Trig}(p,u)} E_u \;\Big\|\; \mathop{\Big\|}_{s \in \mathcal{S}, \mathsf{Trig}(p,s)} \alpha_{p,s} \right)$$

As can be seen, processes now produce a unique atom for all their output synchronisers. To comply with the notational conventions of Process Algebra, we denote atomic actions in lowercase.

It should be noted that some processes do not trigger any other process control objects. In that case, the merge is specified over the empty set. This exception is dealt with by defining the merge over the empty set to be the empty action ε as this is the neutral element for parallel composition. Therefore, if no process control objects x exist, such that $\mathsf{Trig}(p, x)$, the equation for process p reduces to $E_p = \mathsf{Name}(p)$.

For every terminating decision $d \in \mathcal{D}_t$ we have the following equation in Process Algebra:

$$E_d \quad = \quad \sum_{u \in \mathcal{U}, \mathsf{Trig}(d,u)} E_u \; + \sum_{s \in \mathcal{S}, \mathsf{Trig}(d,s)} \alpha_{d,s} \; + \; \varepsilon$$

The execution of a terminating decision involves the choice between one of the process control objects that are output of that decision and termination. Although not very likely from a practical point of view, in theory it is possible that a decision does not have any output triggers. For completeness' sake, we therefore define the sum over the empty set to be δ as this is the neutral element with respect to summation.

The equation for non-terminating decisions is quite similar, except that termination is no option. Hence the substitution of ε by δ:

$$E_d \quad = \quad \sum_{u \in \mathcal{U}, \mathsf{Trig}(d,u)} E_u \; + \sum_{s \in \mathcal{S}, \mathsf{Trig}(d,s)} \alpha_{d,s} \; + \; \delta$$

Note that if a non-terminating decision does not have any output triggers, it will inevitably result in a deadlock. Otherwise, the δ is removed by axiom A6.

The definition of a synchroniser comprises three parts: an equation and definitions of the communication function and the encapsulation set. The equation for a synchroniser $s \in \mathcal{S}$ is similar to the equation for a process, except that no action is needed:

$$E_s \quad = \quad \mathop{\Big\|}_{u \in \mathcal{U}, \mathsf{Trig}(s,u)} E_u \;\; \Big\| \;\; \mathop{\Big\|}_{s' \in \mathcal{S}, \mathsf{Trig}(s,s')} \alpha_{s,s'}$$

The communication function is defined as:

$$\mathop{\Big|}_{x \in \mathcal{X}, \mathsf{Trig}(x,s)} \alpha_{x,s} \quad = \quad E_s$$

From this it is clear that all input process control objects have to be ready before the synchroniser is initiated. It should be noted that the above equation is present for synchronisers that can be triggered by other process control objects only, i.e. $s \in \mathcal{S}$ such that $\{x \in \mathcal{X} \mid \mathsf{Trig}(x, s)\} \neq \varnothing$.

In order to assure that the α's are used to trigger synchronisers only, the encapsulation operator, ∂_H, has to be applied. For each synchroniser $s \in \mathcal{S}$ the associated encapsulation set H_s is defined as:

$$H_s \quad = \quad \{\alpha_{x,s} \mid \mathsf{Trig}(x, s)\}$$

The set of all communication atoms is defined as $H = \bigcup_{s \in \mathcal{S}} H_s$. The whole system is denoted by $E_{\mathcal{W}}$, which has as corresponding equation:

$$E_{\mathcal{W}} = \partial_H \left(\mathop{\Big\|}_{j \in \mathcal{J}} E_j \right)$$

Decomposition names simply correspond to process variables which reflect in their equation that their execution requires all their initial items to be activated in parallel. Hence, for every $V \in \mathsf{ran}(\mathsf{Sup})$:

$$V \;=\; \mathop{\Big\|}_{i \in \mathcal{I}, \mathsf{Sup}(i)=V} E_i$$

Example 2.2 Consider again process control specification $\mathcal{W}$ of figure 2. Application of the previously described translation yields:

$$\begin{array}{lcllcl}
E_{\mathcal{W}} & = & \partial_{\{\alpha_{p_2,s_1},\alpha_{p_3,s_1}\}}(E_{p_1}) & E_{p_1} & = & a \cdot (E_{p_2} \parallel E_{p_3}) \\
E_{p_2} & = & b \cdot (E_{d_1} \parallel \alpha_{p_2,s_1}) & E_{p_3} & = & c \cdot \alpha_{p_3,s_1} \\
E_{d_1} & = & E_{p_4} + \varepsilon & E_{s_1} & = & E_{p_5} \\
E_{p_4} & = & d & E_{p_5} & = & a
\end{array}$$

where

$$\alpha_{p_2,s_1} \mid \alpha_{p_3,s_1} \;=\; \alpha_{p_3,s_1} \mid \alpha_{p_2,s_1} = E_{s_1}$$

The axioms for ACP_ε allow $E_{\mathcal{W}}$ to be rewritten to:

$$E_{\mathcal{W}} = a(b(dca + c(da + a(d+\epsilon))) + cb(da + a + a(d+\epsilon)))$$

□

3 Some Verification Problems and Their Complexity

In this section focus is on the definition of verification problems in process control specifications and their complexity. These problems solely concern process control issues and assume that specifications may use the core process control concepts as presented in the previous section.

In a process control specification it can be relevant to be able to determine whether a certain process can be invoked at all. The *initiation problem* for a process p is to decide whether there is a sequence of events leading to the execution of p. For a formal formulation of the initiation problem it is necessary to label atomic actions with the process that was responsible for their execution as different processes can have identical names and hence invoke the same atomic action. In order to achieve this, the ACP_ε equation for a process $p \in \mathcal{P}$ not having a decomposition (i.e. $\mathsf{Name}(p) \in \mathcal{A}$) is changed to:

$$E_p \;=\; \mathsf{Name}(p)[p] \cdot \left(\mathop{\Big\|}_{u \in \mathcal{U}, \mathsf{Trig}(p,u)} E_u \;\Big\|\; \mathop{\Big\|}_{s \in \mathcal{S}, \mathsf{Trig}(p,s)} \alpha_{p,s} \right)$$

A process p can be invoked iff there is a trace containing the atomic action $\mathsf{Name}(p)[p]$.

Definition 3.1
The initiation problem for a process p in a process control specification $\mathcal{W}$ is to decide whether there exists a $\sigma \in \mathcal{A}^*$ such that $\sigma \cdot \mathsf{Name}(p)[p]$ is a trace of $E_{\mathcal{W}}$. □

Theorem 3.1 The initiation problem is NTIME(poly)-complete for process control specifications without decomposition (i.e. $\mathsf{dom}(\mathsf{Sup}) = \varnothing$).

Proof:

For this proof we describe a polynomial time transformation of SATISFIABILITY (SAT for short) to the initiation problem. The proof given here is inspired by the proof given in [13], where a comparable translation was used for proving the fact that determining liveness in a Free Choice Petri Net is an NP-complete problem.

Let K be an instance of SAT. The process control specification $\mathcal{W}_K$ resulting from the transformation has for each boolean variable $u \in U$ a decision D_u as initial item. This set of decisions is to capture all possible truth assignments to the boolean variables in U. Hence, each decision D_u has an output trigger to a process with name u and an output trigger to a process with name $\sim u$ (to avoid confusion, the symbol $\sim$ is used for logical negation in this proof).

Let p be a process with name x, where x is a literal. For each clause C in K for which $\sim x \in C$ a trigger from p to a process with name $\langle \sim x, C\rangle$ exists (where it is assumed that $\sim\sim x \equiv x$). Further, for each C a synchroniser S_C exists which has as input all processes with names of the form $\langle x, C\rangle$. Finally, each synchroniser S_C has an output trigger to a process with name K. Note that the construction implies that a synchroniser S_C will only be triggered if every one of its input processes will be executed, which means that each of the literals in C evaluates to false in a particular truth assignment.

The construction guarantees that K can be initiated if and only if K is not satisfiable. Suppose K is not satisfiable, then in any truth assignment T there is a clause $C = \{x_1, \ldots, x_n\}$ that evaluates to false under T. Hence, all literals x_i evaluate to false under T. Synchroniser S_C depends on the processes with names $\langle \sim x_1, C\rangle, \ldots, \langle \sim x_n, C\rangle$. All these processes will be started as a process with name $\langle \sim x_i, C\rangle$ will be started by a process with name $\sim x_i$. As x_i evaluates to false under T, decision D_{u_i}, where u_i is the boolean variable in literal x_i, will trigger the process with name $\sim x_i$. Synchroniser S_C will then trigger the process with name K.

This intuitive reasoning can be used as a reduction strategy for the corresponding process algebra terms, to prove that K occurs in a trace of $E_{\mathcal{W}_K}$. As the construction is such that each process has a unique name, we ignore the difference between processes and their names. The following abbreviations for traces are used: $X =\sim x_1 \ldots \sim x_n$ and $Y = \langle \sim x_1, C\rangle, \ldots, \langle \sim x_n, C\rangle$.

$$
\begin{aligned}
E_{\mathcal{W}_K} &= \partial_H\left(\mathop{\Big\|}_{1\le i\le |U|} E_{D_{u_i}}\right)\\
&= \partial_H\left(\mathop{\Big\|}_{1\le i\le n}\left(u_i \mathop{\|}_{\substack{d\in K,\\ \sim u_i \in d}} \langle u_i, d\rangle\, \alpha_{\langle u_i,d\rangle,S_d} + \sim u_i \mathop{\|}_{\substack{d\in K,\\ u_i \in d}} \langle \sim u_i, d\rangle\, \alpha_{\langle \sim u_i,d\rangle,S_d}\right) \mathop{\Big\|}_{n+1\le j\le |U|} E_{D_{u_j}}\right)\\
&\overset{X}{\twoheadrightarrow} \partial_H\left(\mathop{\Big\|}_{1\le i\le n}\left(\mathop{\|}_{\substack{d\in K,\\ x_i \in d}} \langle \sim x_i, d\rangle\, \alpha_{\langle \sim x_i,d\rangle,S_d}\right) \mathop{\Big\|}_{n+1\le j\le |U|} E_{D_{u_j}}\right)
\end{aligned}
$$

$$
\begin{aligned}
\overset{XY}{\twoheadrightarrow}\quad & \partial_H\Bigg(\mathop{\Big\|}_{1\le i\le n}\Big(\alpha_{\langle\sim x_i,C\rangle,S_C}\mathop{\|}_{\substack{d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\Big)\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)\\
=\quad & \partial_H\Bigg(\Big(\mathop{\Big\|}_{1\le i\le n}\alpha_{\langle\sim x_i,C\rangle,S_C}\Big)\mathop{\|}_{\substack{1\,\le\,i\,\le\,n,\\ d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)\\
=\quad & \partial_H\Bigg(\Big(\mathop{\Big|}_{1\le i\le n}\alpha_{\langle\sim x_i,C\rangle,S_C}\Big)\mathop{\|}_{\substack{1\,\le\,i\,\le\,n,\\ d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)\\
=\quad & \partial_H\Bigg((E_{S_C})\mathop{\|}_{\substack{1\,\le\,i\,\le\,n,\\ d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)\\
=\quad & \partial_H\Bigg((K)\mathop{\|}_{\substack{1\,\le\,i\,\le\,n,\\ d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)\\
\overset{XYK}{\twoheadrightarrow}\quad & \partial_H\Bigg(\mathop{\|}_{\substack{1\,\le\,i\,\le\,n,\\ d\,\in\,K\setminus\{C\},\\ x_i\,\in\,d}}\langle\sim x_i,d\rangle\,\alpha_{\langle\sim x_i,d\rangle,S_d}\mathop{\Big\|}_{n+1\le j\le|U|}E_{D_{u_j}}\Bigg)
\end{aligned}
$$

Hence K occurs in a trace of $E_{\mathcal{W}_K}$.

By analogous reasoning, if K is satisfiable, then none of the synchronisers S_C can be started and K will not occur in a trace of $E_{\mathcal{W}_K}$.

Clearly this construction is a polynomial time transformation. Finally, to prove that the problem is in NP, it is sufficient to observe that the verification of an execution scenario can be done in polynomial time. Note that the translation will never result in loops and that decomposition is not used. □

Example 3.1 The construction of the previous proof is illustrated in figure 3, where the result of the translation of the formula $K = A \wedge B \wedge C = (p \vee \sim q \vee r) \wedge (p \vee \sim r) \wedge (q \vee \sim r)$ to a process control specification is presented. □

For some process control specifications termination is essential; e.g. for workflows as they basically correspond to long term transactions. For other process control specifications nontermination is crucial, as they are to represent continuous processes (e.g. an operating system). The question whether a process control specification can terminate is called the *termination problem*.

Definition 3.2

A process control specification $\mathcal{W}$ can reach a terminal state iff $E_{\mathcal{W}} \twoheadrightarrow \surd$. □

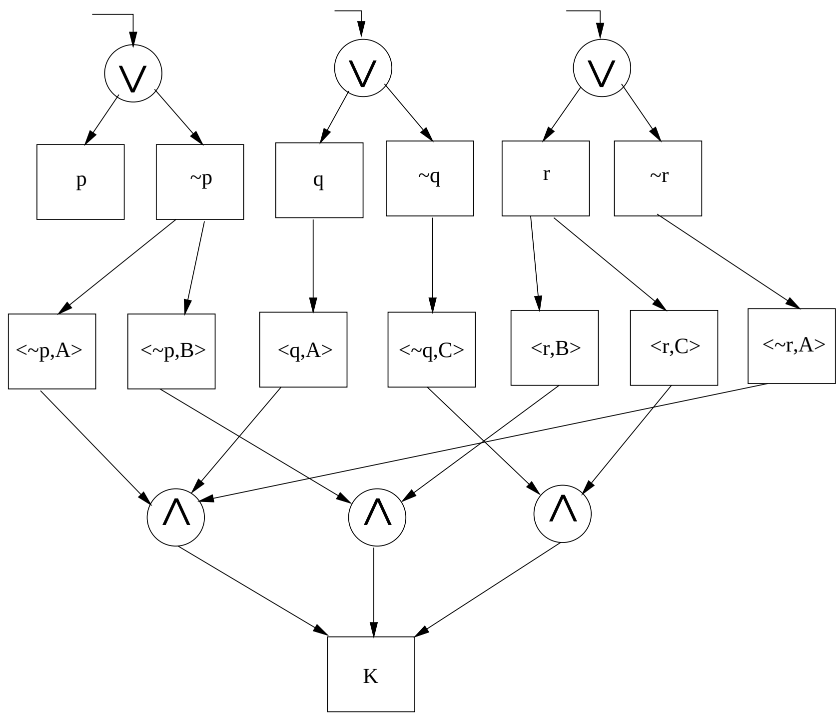

Figure 3: Process control specification for $K = A \wedge B \wedge C = (p \vee \sim q \vee r) \wedge (p \vee \sim r) \wedge (q \vee \sim r)$

Theorem 3.2 The terminal state reachability problem for process control specifications without decomposition is DSPACE(exp)-hard.

Proof:

By a reduction of the zero-reachability problem for ordinary Petri Nets to the terminal state reachability problem. An ordinary Petri Net is a four tuple $\langle P, T, I, O \rangle$, where P is a set of places, T is a set of transitions, $I: T \to \wp(P)$ is the input function, a mapping from transitions to sets of places, and $O: T \to \wp(P)$ is the output function, also a mapping from transitions to sets of places. The zero-reachability problem is to determine whether a marking μ' exists, reachable from the initial marking μ, such that $\mu'(p_i) = 0$ for all $p_i \in P$. As shown in [19], 1) the reachability problem for ordinary Petri Nets is equivalent to the reachability problem for general Petri Nets and 2) the reachability problem is reducible to the zero-reachability problem. In [13] it is proven that the reachability problem is DSPACE(exp)-hard (an algorithm can be found in [16]).

Let $\mathcal{C} = \langle P, T, I, O \rangle$ be an ordinary Petri Net with initial marking μ. The corresponding process control specification $\mathcal{W}_{\mathcal{C}_\mu}$ has a process T_p for each place $p \in P$ and a synchroniser S_t for each transition $t \in T$. For each place $p \in P$ a non-terminating decision D_p is introduced and a trigger from T_p to D_p as well as triggers from D_p to S_t for each transition t that has p as input, i.e. for each $t \in I(p)$. Finally, each arrow from a transition t to a place p results in a trigger from S_t to T_p. Transitions without input places will correspond to synchronisers, which are initial item and have triggers which have that synchronisers both as source and as target. Transitions without input places are special, as they can keep generating tokens for their output place. In the corresponding process control specification this behaviour is simulated by having the corresponding synchroniser always active (achieved by having it as initial item and assigning it a "self" trigger).

Apart from the transitions without input places, the initial marking μ determines the initial items of the process control specification $\mathcal{W}_{\mathcal{C}_\mu}$, For each place p with n tokens in marking μ ($n > 0$), we create n synchronisers I_p^i, without input triggers, but with exactly one output trigger to process T_p. All such synchronisers are initial item of $\mathcal{W}_{\mathcal{C}_\mu}$. Note that because of the fact that the trigger relation Trig is a set, we cannot create a single synchroniser for p with n arrows to process T_p.

To prove the theorem we now need to prove that the terminal state in $\mathcal{W}$ can be reached if and only if a zero marking μ' exists for $\mathcal{W}_{\mathcal{C}_\mu}$.

To prove that if a zero marking μ' exists for $\mathcal{C}$ a terminal state can be reached in $\mathcal{W}_{\mathcal{C}_\mu}$, we prove that if a marking ν can be reached in $\mathcal{C}$, via a sequence of transitions $t_0, \ldots t_n$, $E_{\mathcal{W}_{\mathcal{C}_\mu}}$ can be rewritten to $\partial_H(\underset{p\in P,\nu(p)>0}{\|} E_{T_p}^{\nu(p)} \underset{t\in T,I(t)=\varnothing}{\|} E_{S_t})$. For the initial state μ this is trivial:

$$E_{\mathcal{W}_{\mathcal{C}_\mu}} = \partial_H\left(\underset{p\in P,1\leq i\leq \mu(p)}{\|} E_{I_p^i} \underset{t\in T,I(t)=\varnothing}{\|} E_{S_t}\right) = \partial_H\left(\underset{p\in P,\mu(p)>0}{\|} E_{T_p}^{\mu(p)} \underset{t\in T,I(t)=\varnothing}{\|} E_{S_t}\right)$$

Consider the marking ν reached after firing transition t_m, using induction we have that $E_{\mathcal{W}}$ can be reduced to

$$\partial_H\left(\underset{p\in P,\nu(p)>0}{\|} E_{T_p}^{\nu(p)} \underset{t\in T,I(t)=\varnothing}{\|} E_{S_t}\right)$$

Firing t_{m+1}, assuming $I(t_{m+1}) \neq \varnothing$, resulting in a marking $\delta(\nu, t_{m+1})$ corresponds in ACP-terms to:

$$\begin{aligned}
E_{\mathcal{W}_{\mathcal{C}_\mu}} &= \partial_H\left(\underset{p\in P,\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&= \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1})}{\Big\|} E_{T_p}^{\nu(p)} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&= \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1}),\nu(p)>1}{\Big\|} E_{T_p}^{\nu(p)-1} \underset{p\in I(t_{m+1})}{\Big\|} E_{T_p} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&= \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1}),\nu(p)>1}{\Big\|} E_{T_p}^{\nu(p)-1} \underset{p\in I(t_{m+1})}{\Big\|} p\cdot E_{D_p} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&\twoheadrightarrow \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1}),\nu(p)>1}{\Big\|} E_{T_p}^{\nu(p)-1} \underset{p\in I(t_{m+1})}{\Big\|} \alpha_{p,S_{t_{m+1}}} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&= \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1}),\nu(p)>1}{\Big\|} E_{T_p}^{\nu(p)-1} \underset{p\in I(t_{m+1})}{\Big|} \alpha_{p,S_{t_{m+1}}} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right) \\
&= \partial_H\left(\underset{p\in P\setminus I(t_{m+1}),\nu(p)>0}{\Big\|} E_{T_p}^{\nu(p)} \underset{p\in I(t_{m+1}),\nu(p)>1}{\Big\|} E_{T_p}^{\nu(p)-1} \Big\| E_{S_{t_{m+1}}} \underset{t\in T,I(t)=\varnothing}{\Big\|} E_{S_t}\right)
\end{aligned}$$

$$= \partial_H \left(\mathop{\|}_{p \in P \setminus I(t_{m+1}), \nu(p) > 0} E_{T_p}^{\nu(p)} \mathop{\|}_{p \in I(t_{m+1}), \nu(p) > 1} E_{T_p}^{\nu(p)-1} \mathop{\|}_{q \in O(t_{m+1})} E_{T_q} \mathop{\|}_{t \in T, I(t) = \varnothing} E_{S_t} \right)$$

$$= \partial_H \left(\mathop{\|}_{p \in P, \delta(\nu, t_{m+1})(p) > 0} E_{T_p}^{\delta(\nu, t_{m+1})(p)} \mathop{\|}_{t \in T, I(t) = \varnothing} E_{S_t} \right)$$

For t_{m+1} with $I(t_{m+1}) = \varnothing$ the rewriting is very similar.

Firing the last transition t_n leads to the zero marking μ'. This means that this transition cannot have any output places and hence, the corresponding synchronizer S_{t_n} does not have any output processes and its rewrite rule in ACP$_\varepsilon$ simply is: $E_{S_{t_n}} = \varepsilon$. Just before t_n fires, we have a marking μ'' which only has tokens in the input places of t_n (one in each such place). Hence, $\partial_H \left(\mathop{\|}_{p \in I(t_n)} E_{T_p} \right)$ can simply be reduced to ε, which completes this part of the proof.

Proving that if a terminal state can be reached, a zero marking exists follows similar lines and also uses the strict correspondence between the execution of synchronisers and the firing of transitions. □

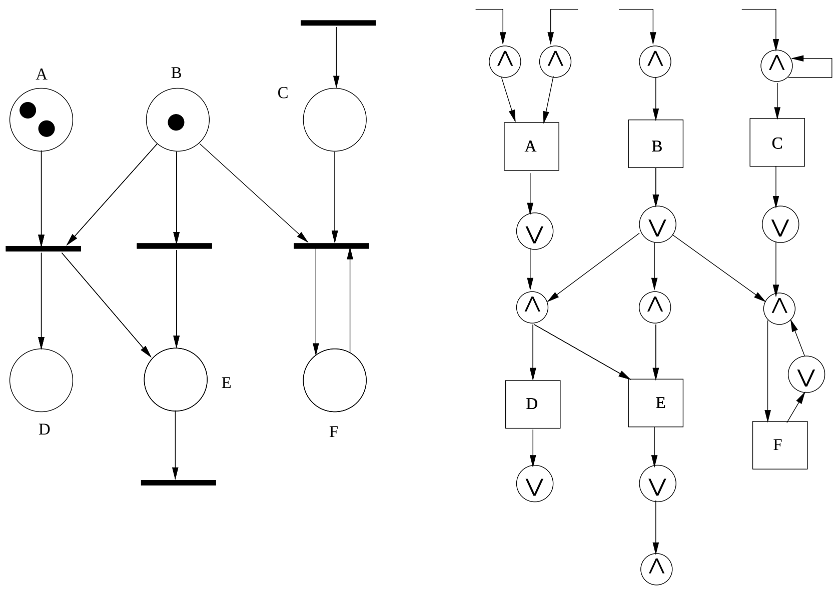

Figure 4: Translating Petri Nets to Process Control Specifications

Example 3.2 In figure 4 an example of a Petri Net with an initial marking and its corresponding process control specification is shown.

□

Corollary 3.1 The initiation problem for process control specifications with decomposition is DSPACE(exp)-hard.

Proof:

Consider a Petri net $\mathcal{C}$ and its translation as described in theorem 3.2. Definine a new process control specification with a process with name M as initial item, followed by a process with name N. Define the decomposition of M to be the result of the translation of $\mathcal{C}$. The process named N can be initiated if and only if the process named M terminates, i.e. if and only if a zero marking can be reached in $\mathcal{C}$. $\Box$

Of course, while a process control specification may reach a terminal state, it may also contain execution paths which, once chosen, can never lead to termination. A simple example would be a process control specification having a terminating decision as initial item and a choice for a process which has an output trigger to itself. The *generalized termination problem* for a process control specification $\mathcal{W}$ is to determine whether from every reachable state, the terminal state can be reached.

Definition 3.3

A process control specification $\mathcal{W}$ can always reach a terminal state iff for every process algebra expression N for which $E_{\mathcal{W}} \twoheadrightarrow N$ we have $N \twoheadrightarrow \surd$. $\Box$

Corollary 3.2 The generalized termination problem is DSPACE(exp)-hard.

Reachability (or "nonreachability") of a terminal state is a rather weak requirement. Typically, process control specifications should be deadlock free.

Definition 3.4

A process control specification $\mathcal{W}$ is deadlock free iff $E_{\mathcal{W}} \not\twoheadrightarrow \delta$. $\Box$

Theorem 3.3 Determining whether a process control specification without decomposition is deadlock free is DSPACE(exp)-hard.

Proof:

A marking of a Petri net is a deadlock if it enables no transitions. The *deadlock problem* for a net is the problem of deciding if any of its reachable markings is a deadlock. In [5] it is proven that the reachability problem can be reduced to the deadlock problem and hence that the deadlock problem is DSPACE(exp)-hard.

The deadlock problem for a Petri net $\mathcal{C}$ can be reduced to the deadlock problem for process control specifications, by using the translation presented in the proof of theorem 3.2. However, a slight modification is needed as the zero marking is considered a deadlock in a Petri net, while a terminal state in a process control specification is not. To deal with this, a new non-terminating decision is introduced in the translation, which is an initial item and has no output triggers (the most compact way of representing a deadlock). This construction guarantees that if the Petri net reaches the zero marking, the corresponding process control specification reaches a deadlock. The reason for this is that the zero marking corresponds to process control specification $\mathcal{W}_{\mathcal{C}_\mu}$, resulting from the translation of $\mathcal{C}$, reaching the terminal state. As $\mathcal{W}_{\mathcal{C}_\mu}$ runs in parallel with deadlock, we have $\epsilon \parallel \delta$ which reduces to δ.

Also, if Petri net $\mathcal{C}$ can reach a deadlock not equal to the zero marking, the corresponding process control specification will contain a deadlock. Two cases need to be distinguished. In the first case, there is a place p which contains tokens, but which is not input for any transition. In the translation this corresponds to a decision D_p not having any output triggers (which will rewrite to δ). In the second case, there is a transition t which has an input place with at least one token, but one or more input places without tokens as well. This corresponds to a synchroniser S_t in the translation for which some, but not all communication actions are present. This also results in a deadlock. Proving that if the process control specification resulting from the translation does not contain a deadlock, Petri net $\mathcal{C}$ does not contain a deadlock is analogous. □

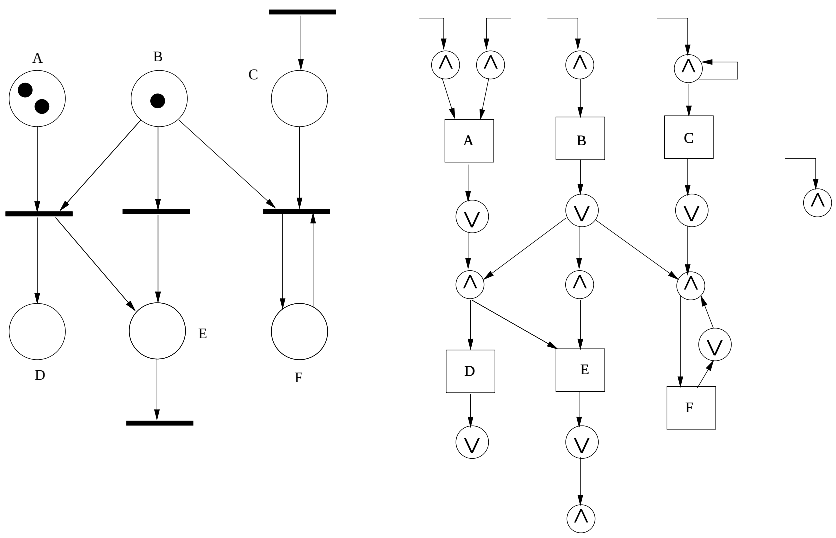

Figure 5: Translating Petri Nets to Process Control Specifications

Example 3.3 In figure 5 an example of a Petri Net with an initial marking and its corresponding process control specification in the context of the deadlock problem is shown. □

Consider a process control specification with two initial items: a non-terminating decision without output triggers and a process named A that runs indefinitely (as it contains an output trigger to itself). This specification is, according to definition 3.3, deadlock free as $\delta \parallel A^\infty = A^\infty$. Although deadlock free, specifications with execution paths which may result in deadlock typically are undesirable.

Definition 3.5

A process control specification $\mathcal{W}$ contains a partial deadlock iff a process control object $x \in \mathcal{X}$ exists such that x can be initiated and $E_x \twoheadrightarrow \delta$. □

Corollary 3.3 Detecting partial deadlocks is DSPACE(exp)-hard.

Process control specifications may be deadlock free, but still suffer from livelock. Examples are cycles of only decisions, a process with a process with the same name as initial item etc. In other words: a state, which is neither deadlock nor terminal, where no atomic actions can be performed.

Definition 3.6

A process control specification $\mathcal{W}$ contains livelock iff $E_{\mathcal{W}} \twoheadrightarrow N$ with $N \neq \delta$ and $N \neq \epsilon$ and no $a \in \mathcal{A}$ exists such that $N \xrightarrow{a} N'$. □

It should be noted that for individual process control objects the detection of livelock is quite straightfoward: the corresponding process algebra equations should be guarded (which can be detected in linear time, as this corresponds to the detection of cycles in a directed graph, see [1]). If every process control object is free of livelock, then obviously the whole specification is too. The reverse, however, is not true and detection of livelock in a process control specification is as complex as detecting deadlock.

Theorem 3.4 Detecting whether a process control specification $\mathcal{W}$ is livelock free is DSPACE(exp)-hard.

Proof:

Consider a Petri net $\mathcal{C}$ and its translation as described in the proof of theorem 3.3. We extend this translation, to a process control specification $L_{\mathcal{C}}$, by simply adding an initial item, a decision D which has an output trigger to itself and no other triggers. Clearly, D represents a livelock. The translation described in the proof of theorem 3.3 can never result in a livelock as it is easily verified that every process control object is livelock free. Its construction guarantees that it can also never terminate. Hence, there are two options: it may contain a deadlock, or it executes indefinitely. If it contains a deadlock, $L_{\mathcal{C}}$ contains a livelock as $\delta \parallel E_D$ represents a livelock. It is executes indefinitely, $L_{\mathcal{C}}$ does not contain a livelock as actions can always be performed. Hence, we have shown that $L_{\mathcal{C}}$ contains a livelock if and only if $\mathcal{C}$ contains a deadlock. □

Example 3.4 Figure 6 illustrates the previous proof. □

Often it is desirable to determine whether two specifications are equivalent. This might be relevant e.g. in the context of execution optimization. Here we adopt a relatively straightforward notion of equivalence: *trace equivalence.*

Definition 3.7

Process control specifications $\mathcal{W}$ and $\mathcal{V}$ are equivalent iff their trace sets are equal.

□

This definition provides perhaps a relatively simple look on equivalence as it assumes that, in Process Algebra terms, $X(Y + Z) = XY + XZ$. As we do not consider the relation between process control specifications and data, this is only natural: decisions cannot be influenced by data.

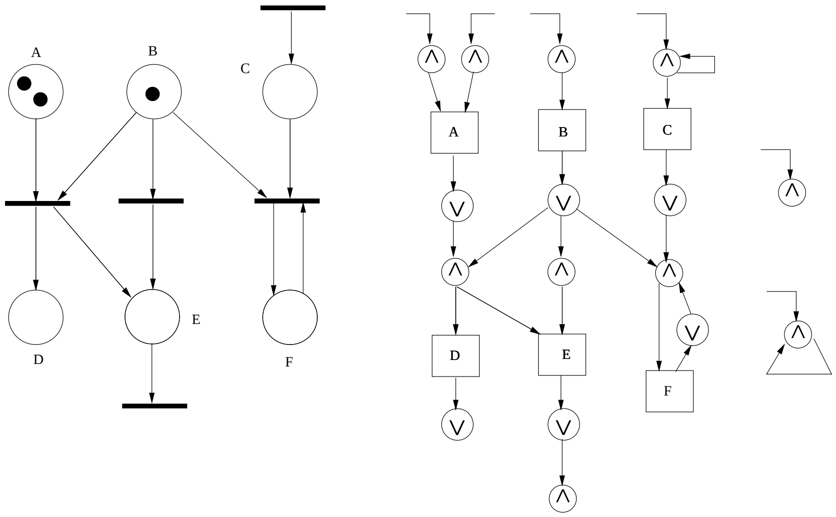

Figure 6: Detecting livelock in process control specifications

Theorem 3.5 Equivalence for process control specifications without decomposition is undecidable.

Proof:

Follows immediately from the translation of Ordinary Petri Nets to process control specifications given in the proof of theorem 3.2 and the fact that equivalence for Ordinary Petri Nets is undecidable. □

Some techniques use restricted forms of synchronisation, as they do not have a synchroniser concept, but use decomposition to capture synchronisation (this is e.g. the case for the Task Structures of [22, 2, 7, 3]). A decomposition terminates iff all its execution paths have terminated. Not all forms of synchronisation however can be captured by superimposing a certain decomposition (the process control specification of figure 2 provides an example).

Definition 3.8

A restricted *process control specification is a process control specification without synchronisers, i.e.* $\mathcal{S} = \varnothing$*.*

□

Even for these type of process control specifications, equivalence is undecidable.

Theorem 3.6 The equivalence problem for restricted process control structures is undecidable.

Proof:

For context-free grammars G_1 and G_2 to determine whether $L(G_1) = L(G_2)$ is undecidable (see e.g. [20]). Formally, a context-free grammar G is a tuple $\langle N, \Sigma, \Pi, S\rangle$, where N

is a finite set of nonterminal symbols, Σ is a finite set of terminal symbols, $S \in N$ is the initial symbol and Π is a set of production rules of the form $A \to \omega$ where $A \in N$ and $\omega \in (N \cup \Sigma)^*$.

A context-free grammar G can be translated to a process control specification $\mathcal{W}_G$ such that $w \in L(G)$ if and only if w is a trace of $\mathcal{W}_G$. For every nonterminal t there will be a decomposition with name t, having a decision D_t as initial item. This decision is terminating iff G contains a rule of the form $t \to \varepsilon$. Let P be a production rule of the form $t \to s_1, \ldots, s_n$. For every s_i $(1 \leq i \leq n)$ there is a process T_{P_i} with name s_i in the decomposition of name t. Furthermore, there is a trigger from D_t to the process with name s_1. Hence production rule P results in a sequence of processes in an order corresponding to the order of the nonterminals and terminals s_i in its righthand side. As P might be one of more production rules for t, decision D_t is introduced allowing a choice for P. The main process has a decision D_S as initial item (recall that S is the initial symbol of the grammar). This completes the construction of $\mathcal{W}_G$.

If $w \in L(G)$, then the corresponding trace simply follows a (!) derivation of w by choosing in each decision associated with each nonterminal the production rule chosen in that derivation. This follows trivially from the fact that for every nonterminal N the equation for the corresponding Process Algebra variable is:

$$N = E_{D_N} = \sum_{\substack{P \in \Pi, \\ P : N \to w_1 \ldots w_n}} E_{T_P} = \sum_{\substack{P \in \Pi, \\ P : N \to w_1 \ldots w_n}} E_{T_{w_1}} \ldots E_{T_{w_n}}$$

The other way around is identical. Hence, we achieved a one-to-one correspondence between context-free grammars and process control specifcations.

□

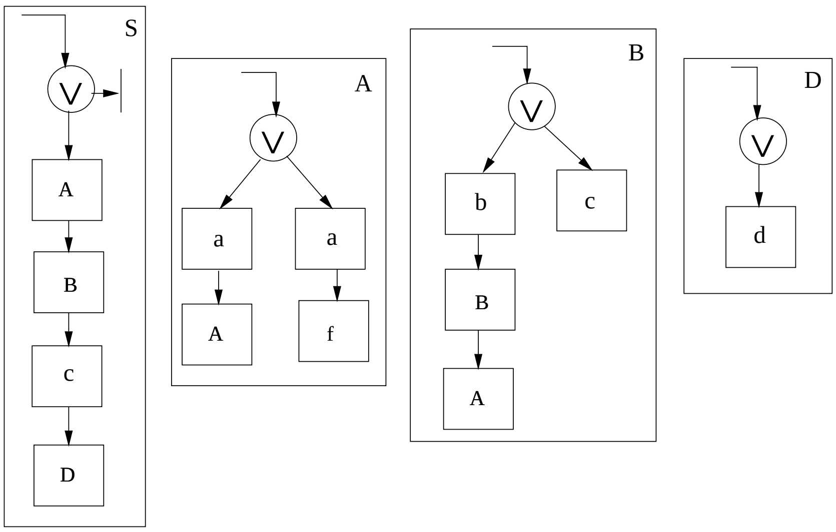

Figure 7: Process control specification for context-free grammar G

Example 3.5 In figure 7, the process control specification $\mathcal{W}_G$ for a context-free grammar G is given. This grammar is defined by the following production rules:

$$\begin{array}{lllllll} S & \rightarrow & ABcD & \quad & S & \rightarrow & \varepsilon \\ A & \rightarrow & aA & & A & \rightarrow & af \\ B & \rightarrow & bBA & & B & \rightarrow & c \\ D & \rightarrow & d & & & & \end{array}$$

□

Theorem 3.6 implies that we have a situation similar to query optimization. Equivalence of first order queries for example is not decidable, hence focus is on the application of equivalence-preserving transformations. For process control specifications, also desirable transformations could be defined, supporting the "optimization" of their execution.

The fact that restricted process control specifications can capture any context-free grammar has an interesting consequence.

Lemma 3.1 Process control specifications are more expressive than Petri nets.

Proof:

Follows immediately from the proofs of theorems 3.6 and 3.2 and theorem 6-9 on page 179 of [19], which states that there are context-free languages which are not Petri net languages. □

Of course this raises further questions with respect to the expressive power of process control specifications as defined in this paper. As a proper investigation of their expressive power is outside the scope of this paper, we leave it as a topic for further research, and mention the following result only.

Lemma 3.2 The language of process control specifications is more powerful than the union of context-free languages and Petri net languages.

Proof:

The proofs of theorems 3.2 and 3.6 show that process control specifications are more powerful than Petri net languages and context-free languages respectively. As proven in [19] the language $\{ww^r \mid w \in \Sigma^*\}$ is not a Petri net language ($|\Sigma| > 1$). As it is context-free however, it can be represented as a process control specification (say P). The languages $a^n b^n c^n$ on the other hand is not context-free, but can be captured as a Petri net. Let Q be the process control specification capturing this language. The language $\{ww^r \mid w \in \Sigma^*\} \parallel a^n b^n c^n$ is neither a context-free language nor a Petri net language, but is captured by the process control specification having P and Q run in parallel. □

4 Conclusions

This paper concentrated on verification problems and their computational complexity in process control specifications. Typical verification problems dealt with the detection of unreachable process control objects, infinite loops, deadlock and livelock. As was shown, most of these problems have a computational complexity which is unacceptable from a practical point of view. It should be remarked, however, that this complexity pertains to worst case situations. In certain cases, the average case complexity may be perfectly acceptable (this also strongly depends on the type of application).

In general, in order to overcome the problems associated with the high complexity of verification problems in process control specifications, further research may consider focussing on heuristics and incremental algorithms on the one hand and syntactic restrictions on the other hand. This latter direction is a direction that has led to the development of many types of Petri nets.

Acknowledgements

The authors would like to thank David Truffet for his insightful suggestions and constructive criticisms and Colin Fidge for carefully reading and commenting upon an earlier version of this paper.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge University Press, Cambridge, United Kingdom, 1990.

[2] P.W.G. Bots. An Environment to Support Problem Solving. *Decision Support Systems*, 3(3):225–231, December 1987.

[3] P.W.G. Bots. *An Environment to Support Problem Solving.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 1989.

[4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual Modeling of Workflows. In M.P. Papazoglou, editor, *Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer-Verlag, December 1995.

[5] A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147:117–136, 1995.

[6] B. Curtis, M.I. Keller, and J. Over. Process Modelling. *Communications of the ACM*, 35:75–90, September 1992.

[7] R.C.J. Dur and P.W.G. Bots. Dynamic Modelling of Organizations Using Task/Actor Simulation. In R.L. Crosslin and H.G. Sol, editors, *Proceedings of the second International Working Conference on Dynamic Modelling of Information Systems*, pages 49–71, Amsterdam, The Netherlands, 1991. Elsevier Science Publishers.

[8] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, May 1996.

[9] J.A. Gulla, O.I. Lindland, and G. Willumsen. PPP: An Integrated CASE environment. In R. Andersen, J.A. Bubenko, and A. Sølvberg, editors, *Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 194–221, Trondheim, Norway, May 1991. Springer-Verlag.

[10] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[11] A.H.M. ter Hofstede and E.R. Nieuwland. Task structure semantics through process algebra. *Software Engineering Journal*, 8(1):14–20, January 1993.

[12] I. Jacobson, M. Christerson, M. Jonsson, and P. van Overgaard. *OO Software Engineering, A Use Case Driven Approach.* Addison-Wesley, Reading, Massachusets, 1992.

[13] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of Some Problems in Petri Nets. *Theoretical Computer Science*, 4:277–299, 1977.

[14] G. Kristen. *Object Orientation, the KISS Method: From Information Architecture to Information System.* Addison-Wesley, Reading, Massachusetts, 1994.

[15] P. Marttiin. Towards Flexible Process Support with a CASE Shell. In G. Wijers, S. Brinkkemper, and I. Wasserman, editors, *Proceedings of the Sixth International Conference CAiSE'94 on Advanced Information Systems Engineering*, volume 811 of *Lecture Notes in Computer Science*, pages 14–27, Utrecht, The Netherlands, June 1994. Springer-Verlag.

[16] E.W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal of Computing*, 13(3):441–460, 1984.

[17] R. Milner. *Communication and Concurrency.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[18] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, pages 2–13, Monterey, California, 1987.

[19] J.L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, Englewoods Cliffs, New Jersey, 1981.

[20] A. Salomaa. *Formal Languages.* ACM Monograph Series. Academic Press, New York, New York, 1973.

[21] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, 1990. Springer-Verlag.

[22] G.M. Wijers, A.H.M. ter Hofstede, and N.E. van Oosterom. Representation of Information Modelling Knowledge. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 167–223. IOS Press, 1992.