On the Benefits of Rewrite Logic as a Semantics for Algebraic Petri Nets in Computing Siphons and Traps

Nasreddine Aoumeur[1] Gunter Saake

ITI, FIN, Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39016 Magdeburg, Germany
E-mail: {aoumeur|saake}@iti.cs.uni-magdeburg.de

June 2001

[1]This work is supported by a DAAD scholarship.

Abstract

A lot of research is being done to directly apply to high level nets structural techniques similar to those existing for place transition nets. In particular, the question of computing siphons for high level nets is of intrest since the well known results about relations between liveness of a net and control of its siphons. In this paper, we show how one can derive an efficient computing of siphons containing a given place from an appropriate interpretation of algebraic Petri nets in rewrite logic as a true concurrent semantics. Within this method, the verification procedure of symbolic siphons is reduced to two easy-understandable inference rewrite rules instead to a completeness problem as in the recent approach of K. Schmidt (97). Moreover, the proposed method can be implemented using rewriting techniques.

Contents

Chapter 1

Introduction

High level Petri nets, in particular algebraic Petri nets as defined in [Rei91], are widely accepted in specifying complex concurrent systems [JR91]. However, verification of system properties still remains one of the weak side when using algebraic Petri nets. This unsatisfactory state of affair is of course due to the limited number of *effective*, *efficient* and *simple* techniques for the analysis of their structure and behaviour.

The present paper focuses on the benefits of rewriting logic [Mes92] for deriving more efficient yet simple analysis techniques from existing ones. We achieve this goal, first, through an appropriate interpretation of algebraic Petri nets behaviour in rewrite logic. Second, instead of performing the analysis directly using the net definition—as done in most of existing analysis approaches—, we propose rather to reason on the resulting rewrite rules through the introduction of adequate inference rules.

As a first step towards a more general framework for algebraic Petri nets[1] analysis using rewrite logic, we focus our ideas in this paper on the verification / computation of siphons (and traps) in algebraic Petri nets. Where, first, we propose to interpret the behaviour of a given algebraic Petri nets in a form of rewrite rules adopted from the work in [BMSB93] with slight adaptation to our purpose. Second, we propose an adequate data structure –as a tuple– for computing (and verifying) the different siphons in the net on the basis of two simple inference rules with a very easy-understandable associated strategy. As main advantages of the proposed verification of siphons, we cite particularly:

- By manipulating only the net rewrite rules and the two inference rules, the method does not resort to any complex representation—like matrix—of the net or to its unfolding. In this sense the level of abstraction and declarativity of the algebraic Petri net in fully maintained.
- The possibility of directly implementing the method using recent implementation of the MAUDE language [CDE+99].
- Compared to the similar technique proposed in [Sch96, Sch97] is the needless for separately computing all—time and space consuming—input and output unifiers as well as the test for their inclusion.

The rest of this paper is organized as follows. The second section recalls some notions about algebraic Petri nets. In the third section, after recalling some notions about rewriting

[1] And object Petri nets as a particular instance of them.

logic, we present a tailored—to our purpose—interpretation of algebraic Petri nets in this logic. In the third section and it order to emphasize the complexity of existing methods for computing siphons and traps, but also to situate our method with respect to it, in the fourth section we recall the main steps of this method. In the fourth section, we propose a deductive system based on rewrite logic that allows for verifying and computing siphons in very comprehensible way using an appropriate strategy, we briefly compare it with the mentioned method, and finally we present how all the different steps may be used as a MAUDE program, for computing the siphons using current implementation of this language. In the last section, we close this paper by some remarks with outlying our future work.

1.1 Algebraic Petri nets : Some basic definitions

Some definitions about algebraic Petri nets and related used concepts are reviewed; however due to space limitation we assume familiarity with some algebraic concepts like algebras, substitution, unification, etc. More detail about algebraic specifications and algebraic Petri nets can be found particularly in [EM85] and [Rei91] respectively.

Definition 1.1.1 **A signature** $\Sigma = [S, \Omega]$ *consists of a set* S *of* **sorts** *and a family* $\Omega = \{\Omega_{\omega,s}\}_{\omega \in S^*, s \in S}$ *of* **operation symbols**. $\Omega_{\epsilon,s}$ *is the set of* **constant symbols** *of sort* s. *A set of* $\Sigma-$**variables** *is a family* $X = \{X_s\}_{s \in S}$ *of* **variables**. *The set* $T_{\Omega,s}(X)$ *of* $\Omega, X-$**terms of sort** s *is inductively defined by 1.)* $X_s \cup \Omega_{\epsilon,s} \subseteq T_{\Omega,s}(X)$ *and 2.) for* $\omega \in \Omega_{s_1..s_n,s}$ *and* $T_i \in T_{\Omega,s_i}(X), \omega(T_1, .., T_n) \in T_{\Omega,s}(X)$. *The set* $T_{\Omega,s} := T_{\Omega,s}(\emptyset)$ *contains the* **ground terms of sort** s. $T_\Omega(X) := \bigcup_{s \in S} T_{\Omega,s}(X)$ *is the set of* $\Sigma-$**terms over** X. *A* $\Sigma-$**equation** *of sort* s *over* X *is a pair* $[L, R]$ *of terms* L,$R \in T_{\Omega,s}(X)$. **A specification** $D = [\Sigma, E]$ *consists of a signature* Σ *and a set of* E *of* $\Sigma-$*equations.*

In the rest of this paper, we assume that each sort contains at least one constant symbols. Also, we assume that the set of opeartions symbols is composed from constructs and defined ones. In other words, we assume that the quotient terms algebras [EM85] exists, and it is computable modulo the set of equations—which can with these conditions results in a complete rewrite systems [DJ90].

Definition 1.1.2 **Algebraic Net** *A tuple* $AN = [\Sigma; P, T, F; \psi, \xi, \lambda, m_0]$ *is an* **algebraic Petri net** *iff 1.)* $\Sigma = [S, \Omega]$ *is an algebraic specification; 2.) [P,T,F] is a* **net**, *i.e.* P *and* T *are finite and disjoint sets called* **places** *and* **transitions**, *respectively, and* F *is a relation* $F \subseteq (P \times T) \cup (T \times P)$, *the elements of which are called* **arcs**; *3.)* ϕ *is a* **sort assignment** $\psi : P \to S$; *4.)* ξ *assigns a set of* $\Sigma-$*variables* $\xi(t)$ *to each transition* $t \in T$; *5.)* λ *is the* **arc inscription** *such that for* $f = [p, t]$ *or* $f = [t, p] \in F$, $\lambda(f)$ *is a multi-term over* $T_{\Omega,\psi(p)}(\xi(t))$; *6.)* m_0 *is a* **marking**, *i.e. it assigns a finite multi-term over* $T_{\Omega,\psi(p)}(\emptyset)$ *to every* $p \in P$. m_0 *is called the* **initial marking**.

Example 1.1.3 *We use throughout this paper the database maintenance example detailed in [Sch96]. While the net is depicted in figure 1.1, the associated algebraic specification of different tokens is specified using, for instance, an OBJ like language [GWM+92] as follows.*

```
obj Database-M is.
  sorts Reading, Writing, Dot .
    op A, B, C : ⟶ reading .
    op D, E, F : ⟶ writing .
    op • :⟶ Dot.
endo
```

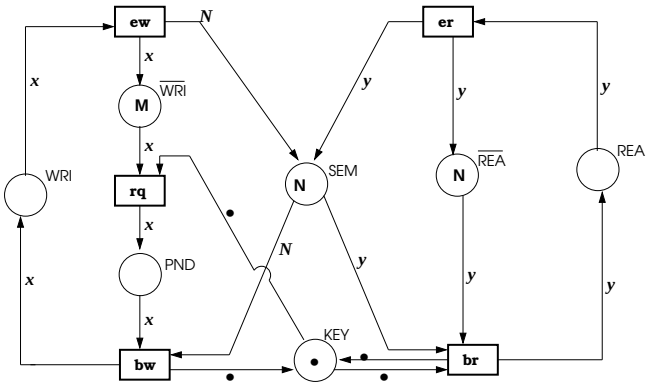

Figure 1.1: The Meta General Forms

Noting that the sorts of places in Figure 1.1 are Writing *for* $WRI, \overline{WRI}$*, and* PND*,* Reading *for* $REA, \overline{REA}$*, and* SEM*, and* Dot *for* KEY*. The letters* N *and* M *are meant to be replaced by the multiterms* $A + B + C$ *and* $D + E + F$*, respectively.*

Finally the place/transition net which underlies an algebraic net is usually constructed as follows:

- The set of places is $\bigcup_{p \in P} \{p\} \times T_{\Omega,\psi(p)}$. As in [Sch97], we write a place $[p, T]$ of the underlying Petri net as $p.T$.
- The set of transitions is $\bigcup_{t \in T} \{p\} \times GrSubst(\phi(t))$[2]. We write a transition $[t, \beta]$ of the underlying Petri net as $t.\beta$.
- The multiplicity of an arc from $p.T$ to $t.\beta$ is given by $(\lambda([p, t])\beta)(T)$. (Note that $\lambda([p, t])\beta$ is a multiset of ground terms.);
- Accordingly the multiplicity of an arc from $t.\beta$ to $p.T$ is $(\lambda([p, t])\beta)(T)$.
- The initial marking of the place $p.T$ is given by $m_0(T)$.

[2]GrSubst denotes all different replacement of variables in $\phi(t)$ by compatible terms with no variables.

Chapter 2

Algebraic Petri nets in rewrite logic

In this section, first we recall some concepts of rewrite logic. Then we present, using the previous example, how the behaviour of a given algebraic Petri net can be specified using this logic. Also, we present how to reason about this behaviour in a true concurrent way. Finally, we give an appropriate interpretation that fits well to our purpose.

2.1 Rewrite logic: An overview

Rewriting logic, considered nowadays as one of the widely accepted unified model of concurrency [Mes98], has been introduced by J. Meseguer in [Mes90], first, by exploiting the intrinsic *concurrency* in terms rewriting [GKM87], and, second, by observing the inadequacy of interpreting rewrite rules as (oriented) equations when dealing with non Platonic (i.e. reactive) systems. While rewrite rules have the usual form, they are rather interpreted in rewriting logic as a *change* in concurrent systems. To be more precise, we present some definitions borrowed from [Mes92]:

- A *(labelled) rewrite theory* $\mathcal{R}$ is a 4-tuple $\mathcal{R} = (\Sigma, E, L, R)$ where Σ is a ranked alphabet of functions symbols, E is a set of Σ-equations, L is a set called the set of *labels,* and R is a set of pairs $R \subseteq L \times (T_{\Sigma,E}(X)^2)^+$ whose first component is a label, and whose second component is a nonempty sequence of pairs of E-equivalence classes of terms, with $X = \{x_1, .., x_n\}$ a countably infinite set of variables. Elements of R are called *rewrite rules.* A rewrite rule $(r, [t], [t'])([u_1], [v_1]). \ . \ .([u_k], [v_k])$ is denoted as

$$r : [t] \Rightarrow [t'] \ if \ [u_1] \Rightarrow [v_1] \wedge ... \wedge [u_k] \Rightarrow [v_k].$$

- Given a rewrite theory $\mathcal{R}$, we say that $\mathcal{R}$ *entails* a sequent $r : [t] \Rightarrow [t']$ and write $\mathcal{R} \vdash [t] \Rightarrow [t']$ if and only if $[t] \Rightarrow [t']$ can be obtained by finite application of the following *rules of deduction*:

  1. **Reflexivity** : For each [t] $\in T_{\Sigma,E}(X)$, $\quad \frac{}{[t] \Rightarrow [t]}$

  2. **Congruence** : For each f $\in \Sigma_n$, n $\in$ N, $\quad \frac{[t_1] \Rightarrow [t'_1]. \ . \ .[t_n] \Rightarrow [t'_n]}{f(t_1,...,t_n)] \Rightarrow [f(t'_1,...,t'_n)]}$

3. **Replacement**[1]: For each rule $r : [t(x_1, .., x_n)] \Rightarrow [t'(x_1, ..., x_n)]$ in R,

$$\frac{[w_1]\Rightarrow[w'_1]...[w_n]\Rightarrow[w'_n]}{[t(\bar{w}/\bar{x})]\Rightarrow[t(\bar{w'}/\bar{x})]}$$

4. **Transitivity** : $\frac{[t_1]\Rightarrow[t_2] \quad [t_2]\Rightarrow[t_3]}{[t_1]\Rightarrow[t_3]}$

- Given a rewrite theory $\mathcal{R}= (\Sigma, E, L, R)$, a (Σ, E)-sequent $[t] \Rightarrow [t']$ is called a *concurrent $\mathcal{R}-$rewrite* (or just a *rewrite*) iff it can be derived from $\mathcal{R}$ by finite application of the rules 1-4. And the sequent is called *one-step concurrent $\mathcal{R}$-rewrite* iff it can be derived from $\mathcal{R}$ by finite application of the rules 1-3 with at least one application of rule 3.

2.2 Algebraic Petri net behaviour in rewrite logic

The interpretation of simple Petri net can be traced back to the first paper that presents rewrite logic [Mes90], where J. Meseguer give a sketch of how reasoning about a given Petri net. The ideas was: first; with each transition an appropriate rewrite rule is associated. Second the (initial) marking is represented as a term composed (using a multiset operator denoted $\otimes$) of the places (with their multiplicity) contaning tokens.

For the algebraic Petri nets, to our knowledge two specific and one more general proposals have been introduced. The two specific ones have been respectively proposed in [BM92, BMSB93] for the ECATNets (algebraic Petri nets) variant and the second was proposed in [BGCM94] as a semantics framework fo the OBJSA variant. The general framework has been recently forwarded by M. Stehr in [Ste98]. In this paper, we follow the first proposal that we adapt afterwards to our purpose.

2.2.1 The Approach in ECATNets

As described in the previous section , the content of each place in algebraic Petri nets is a multiset of ground terms w.r.t. a given signature. The union operation on such multisets is captured using the binary operator denoted $\oplus$. To capture the rewrite rules and such place-states, another multiset form is used where the elements are pairs of the form: *(place_identifier, multisets of terms)*. The union operation on this multiset is denoted by $\otimes$. Lastly, in order to exhibit full concurrency, a distribution law of $\otimes$ over $\oplus$ is defined as an axiom used as 'splitting' from left to right and as 'recombination' from right to left:

$$(P, mt_1 \oplus mt_2) = (P, mt_1) \otimes (P, mt_2)$$

where mt_1 and mt_2 are multisets of terms generated by $\oplus$, while P is a place name. This axiom is referred to as *decomposition* axiom.

Given a transition t, if we denote by $\{P_1, .., P_n\}$ its input places and $\{Q_1, .., Q_m\}$ its output places as depicted in figure 2.1, then the general form of rewrite rule is as follows:

$$T : (P_1, T_1) \otimes (P_2, T_2) \otimes ....(P_n, T_n) \Rightarrow (Q_1, T'_1) \otimes (Q_2, T'_2) \otimes ....(Q_m, T'_m)\ if\ Condition$$

[1] In this rule the term $[t(\bar{w} \ / \ \bar{x})]$ denotes the simultaneous substitution of w_i for x_i in t with $\bar{x}$ representing $(x_1, .., x_n)$.

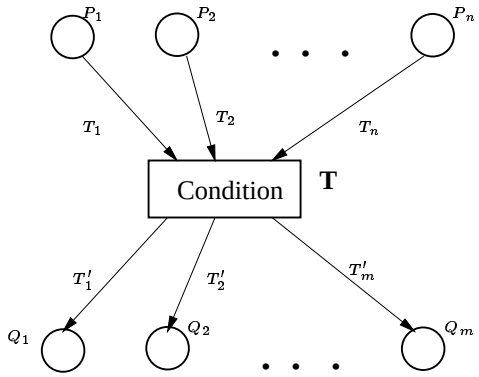

Figure 2.1: Transition general form

Example 2.2.1 *For the net depicted in figure 1.1, by applying the above general form of rewrite rule, we obtain the following rules; one for each transition:*

ew : $(WRI, x) \Rightarrow (\overline{WRI}, x) \otimes (SEM, N)$

rq : $(\overline{WRI}, x) \otimes (KEY, \bullet) \Rightarrow (PND, x)$

bw : $(PND, x) \otimes (SEM, N) \Rightarrow (KEY, \bullet) \otimes (WRI, x)$

er : $(REA, y) \Rightarrow (\overline{REA}, y) \otimes (SEM, y)$

br : $(KEY, \bullet) \otimes (SEM, y) \otimes (\overline{REA}, y) \Rightarrow (KEY, \bullet) \otimes (REA, y)$

The intial state of this net is represented by the following multiset; where all non mentioned places have no marking:

$$(\overline{WRI}, A \oplus B \oplus C) \otimes (SEM, C \oplus D \oplus E) \otimes (\overline{REA}, C \oplus D \oplus E) \otimes (KEY, \bullet)$$

Having this intial state it is quite possible to fire the transition **rq** *or* **br**. *For firing the transition* **rq**, *for instance, we have first to apply the decomposition axiom on the subterm* $(\overline{WRI}, A \oplus B \oplus C)$ *in this initial state. This yields the equivalent subterm* $(\overline{WRI}, A) \otimes (\overline{WRI}, B) \otimes (\overline{WRI}, C)$. *By applying now the commutativity of the multiset operator* $\otimes$, *we result in the following equivalent initial state :*

$$(\overline{\mathbf{WRI}}, \mathbf{A}) \otimes (\mathbf{KEY}, \bullet) \otimes (\overline{WRI}, B) \otimes (\overline{WRI}, C) \otimes (SEM, C \oplus D \oplus E) \otimes (\overline{REA}, C \oplus D \oplus E)$$

So, the rule **rq** *may be applied by substituting the variable x with A. The resulting new state is now (obtained by replacing the instantiated left hand of this rule by the corresponding instantiated right hand side):*

$$(\mathbf{PND}, \mathbf{A}) \otimes (\overline{WRI}, B) \otimes (\overline{WRI}, C) \otimes (SEM, C \oplus D \oplus E) \otimes (\overline{REA}, C \oplus D \oplus E)$$

Noting finally that with this new state, after firing the the transition **bw** *it would be quite possible to firing* **concurrently** *the transitions* **ew** *and* **br**.

2.2.2 More appropriate interpretation

For the verification / compution of siphons that we propose later, a more refined form of rewrite rules is introduced. This new form of rewrite rules is based on the following. First: because our objective is not reasoning about an intial marking (irrelevant for siphons computation) and thus exhibiting a maximal of concurrency, we propose to abstract the operators $\otimes$ and $\oplus$ into a single one that we denote simply by $'$,$'$—which associative and commutative. Second, according to the used notations in the unfolding net instead of the representation $(Place, multiterm)$ we rather use $Place.multiterm$. Third, in coherence with the data structure we propose for computing siphons, we propose to include each side of given rewrite rule between angles as : $\langle ... \rangle$. In this sense and with respect to the generic form of transition in figure 2, each rule now takes rather the following form

$$T : \langle P_1.T_1, ..., P_n.T_n \rangle \Longrightarrow \langle Q_1.T'_1, ..., Q_n.T'_n \rangle$$

Example 2.2.2 *Following this new form, the rewrite rules of our running example are now:*

ew : $\langle WRI.x \rangle \Rightarrow \langle \overline{WRI}.x, SEM.N \rangle$

rq : $\langle \overline{WRI}.x, KEY.\bullet \rangle \Rightarrow \rangle PND.x \rangle$

bw : $\langle PND.x, SEM.N \rangle \Rightarrow \langle KEY.\bullet, WRI.x \rangle$

er : $\langle REA.y \rangle \Rightarrow \langle \overline{REA}.y, SEM.y \rangle$

br : $\langle KEY.\bullet, SEM.y, \overline{REA}.y \rangle \Rightarrow \langle KEY.\bullet, REA.y \rangle$

Chapter 3

An overview of the approach in [Sch96]

The purpose of this section is to overview the main ideas of the method proposed in [Sch96]. It is worth mentioning that for the siphon (or trap) verification of a given user set of symbolic family of places the proposed technique introduces no restriction, whereas for effectively computing siphons the approach has been performed only for unary algebraic Petri nets.

But first, let's recall the definition of a siphon.

Definition 3.0.3 *A subset D of P is a* **siphon** *of a net $N = [P, T, F]$ iff $FD \subseteq DF$ (D is a* **trap** *iff $DF \subseteq FD$). For an algebraic net, siphons and traps are those of the underlying place / transition net.*

As a judicious factorization of different sets of siphons, the approach introduce the notion of *global variable* and the notion of *local variable*. Instantiation of a global variable leads to different set of siphons, while an instantiation of a local variable expand the same set of siphon.

Example 3.0.4 *Using these two factorizations, the author show for instance that the different sets are siphons for the net shown in figure 1.1.*

$\overline{REA}.A$	$REA.A$			
$\overline{REA}.B$	$REA.B$			
$\overline{REA}.C$	$REA.C$			
$\overline{WRI}.D$	$PND.D$	$WRI.D$		
$\overline{WRI}.E$	$PND.E$	$WRI.E$		
$\overline{WRI}.F$	$PND.F$	$WRI.F$		
$WRI.D$	$WRI.E$	$WRI.F$	$SEM.A$	$REA.A$
$WRI.D$	$WRI.E$	$WRI.F$	$SEM.B$	$REA.B$
$WRI.D$	$WRI.E$	$WRI.F$	$SEM.C$	$REA.C$
$PND.D$	$PND.E$	$PND.F$	$KEY.\bullet$	

and they could be simplified to the just four following sets :

$\overline{REA}.X$	$REA.X$	
$\overline{WRI}.Y$	$PND.Y$	$WRI.Y$
$\forall z \in$ Writing $: WRI.z$	$SEM.X$	$REA.X$
$\forall z \in$ Writing $: PND.z$	$KEY.\bullet$	

For capturing this notion of global and local variables, the author proposes the notion of *symbolic family of place sets* as a structure of the form $[P_i.T_i | i \in \Im, X_G, X_L]$, where X_G and X_L are respectively set of local and global variables; P_i is a place and T_i a term of sort P_i and $\Im$ is some set of index. Such family can be reduced to the notion of *symbolic set of places* by instantiating either all local variables X_L or all global variables. From a given symbolic set of places $D = [P_i.T_i | i \in \Im, X_L]$, its *interpretation*, denoted by $UNF(D) = \{p_i.T_i\chi | i \in \Im, \chi \in GrSubst(X_L)\}$, is computed by replacing all local variables in X_L by different (compatible) ground substitutions. The interpretation of a family of set of places is derived similarly.

For the verification-computation of siphons, the proposed approach respects the general procedure proposed for place / transition nets, which can be summarized as follows:

```
var Siphons :  SET OF SET OF places .
(initially empty).

PROCEDURE Siphons(Partial-Siphon, Current-Choice :  SetOfPlaces);

BEGIN .
  FOR EACH p ∈ Current-Choice DO .
    NewPartialSiphon := Partial-Siphon ∪{p};
    IF exists t : t ∈ NewPartialSiphonF \ FNewPartialSiphon THEN;
      Siphons(NewPartialSiphon, tF);
    ELSE
      Siphons := Siphons ∪ {NewPartialSiphon};
    END IF
  END FOR
END siphons.
```

Obviously two main operations have to be performed by this procedure : the check for the siphon property and the expansion of a current set of siphon with new places. Henceforth, the main adaptations proposed in [Sch96] concern a *symbolic* re-implementation of the two operations:

- *Check:* check whether a given symbolically family of places represents siphons or not; if it does, sent it to the output, else send it to the expansion module. If a symbolic representation represents both siphons and non-siphons, split the representation into representations for the siphons and representations of the non-siphons and send the parts to the output or expansion module, respectively.
- *Expand:* for a given symbolic family of places and a transition violating the siphon property, try all possibilities to add an input place of this transition to the place sets and send the results back to the check module.

Noting that as a first adaptation of this general procedure, the approach in [Sch96] starts with $[P.X, \{X\}, \emptyset]$ for all $p \in P$, that is with a symbolic representation of single places, simultaneously for all places. Moreover, at the starting it is assumed that there is only one a global variable and no local variables.

3.1 Checking the siphon property

The proposed verification for a siphon property for a given symbolic family of places $[\mathcal{M}, \mathcal{X}_g, \mathcal{X}_l]$, may be split out in two parts. Verification or test in the general case (step1 until step4) and refinement of this test for the unary free-equation specification case (step 5 until step 11). Hereafter we present an overview just for the first part.

3.1.1 1st step : Computation of input and output unifiers

Let given an algebraic net AN, a transition t, a symbolic family of places $\mathcal{D} = [\{_{\surd}\rangle.\mathcal{T}_{\rangle} \mid \rangle \in \mathcal{I}\}, \mathcal{X}_{\mathcal{G}}, \mathcal{X}_{\mathcal{L}}]$ and some index space I. The set $\mathcal{U}_I(\mathcal{D}, t)$ of **input unifiers** of $\mathcal{D}$ and t is defined by

$$\mathcal{U}_I(\mathcal{D}, t) = \{\rho|_{X_G \cup \xi(t)} | \rho \in GrSubst(X_G \cup X_L \cup \xi(t)) \; and \; \exists \sharp \exists T_I : \sharp \in I, T_I \in \lambda([P_\sharp, t]) \; s.t. \; T_\sharp \rho = T_I \rho\}$$

While the set $\mathcal{U}_O(\mathcal{D}, t)$ of **output unifiers** of $\mathcal{D}$ and t is defined by:

$$\mathcal{U}_O(\mathcal{D}, t) = \{\rho|_{X_G \cup \xi(t)} | \rho \in GrSubst(X_G \cup X_L \cup \xi(t)) \; and \; \exists \flat \exists T_O : \sharp \in I, T_O \in \lambda([t, P_\flat]) \; s.t. \; T_\flat \rho = T_O \rho\}$$

Given such input and output unifiers for each transition and a symbolic family of places $\mathcal{D} = [\{_{\surd}\rangle.\mathcal{T}_{\rangle} \mid \rangle \in \mathcal{I}\}, \mathcal{X}_{\mathcal{G}}, \mathcal{X}_{\mathcal{L}}]$, the Author states the following important result (Lemma 4 in [Sch96]): For every ground term of T of the sort of x_G, the interpretation $UNF(\mathcal{D}[x_G \longrightarrow T])$ is a siphon if an only if for all transitions t of AN the following inclusion holds:

$$\mathcal{U}_O(\mathcal{D}, t)|_{[x_G \longrightarrow T]} \subseteq \mathcal{U}_I(\mathcal{D}, t)|_{[x_G \longrightarrow T]}$$

Where $M|_{[x_G \longrightarrow T]}$ stands for the set of all elements in M which map x to T. This is under the usual assumption that all variables are disjoint (i.e. local, global and transition ones)—that can be achieved through variable renaming.

3.1.2 2sd step : Switch to most general unifiers

This second step consists in keeping free (uninstantiated) the local variables and then apply a factorization for the local variables. This leads to the notion of most general input and output unifiers defined as follows. Always w.r.t. a given algebraic net AN, a transition t, a symbolic family of places $\mathcal{D} = [\{_{\surd}\rangle.\mathcal{T}_{\rangle} \mid \rangle \in \mathcal{I}\}, \mathcal{X}_{\mathcal{G}}, \mathcal{X}_{\mathcal{L}}]$ and some index space I. The set $\mathcal{C}_I(\mathcal{D}, t)$ of **most general input unifiers** of $\mathcal{D}$ and t is defined by:

$$\mathcal{C}_I(\mathcal{D}, t) = \{MGU^1(T_\sharp, T_I)|_{X_G \cup \xi(t)} \mid \sharp \in I, T_I \in \lambda([t, p_\sharp])\}$$

while the set $\mathcal{C}_I(\mathcal{D}, \sqcup)$ of **most general output unifiers** of $\mathcal{D}$ and t is defined by:

$$\mathcal{C}_O(\mathcal{D}, t) = \{MGU(T_\flat, T_O)|_{X_G \cup \xi(t)} \mid \flat \in I, T_O \in \lambda([t, p_\flat])\}$$

The relation between a given $\mathcal{C}_O(\mathcal{D}, t)$ (resp. $\mathcal{C}_I(\mathcal{D}, t)$) and the corresponding $\mathcal{U}_O(\mathcal{D}, t)$ (resp. $\mathcal{U}_I(\mathcal{D}, t)$) is straightforwardly stated (as corollary 2).

[1] With MGU denoting the most greater unifier of the concerned terms.

3.1.3 3rd step : Simplify siphon property to single output unifier

This step stems from the fact that the set of ground substitution of the most general output unifier, namely $Ground(C_o(\mathcal{D},t))$ can be written as as a union of its elementary substitutions, namely $\bigcup_{\sigma_o \in \mathcal{C}_O(\mathcal{D},t)} Ground(\{\sigma_o\})$. Henceforth the global inclusion, namely $Ground(\mathcal{C}_O(\mathcal{D},t))|_{[x_G \longrightarrow T]} \subseteq Ground(\mathcal{C}_I(\mathcal{D},t))|_{[x_G \longrightarrow T]}$, can be reduced to the test of inclusion of each elementary substitution separately. In other words, we have rather to test the inclusion $Ground(\sigma_o)|_{[x_G \longrightarrow T]} \subseteq Ground(\mathcal{C}_I(\mathcal{D},t))|_{[x_G \longrightarrow T]}$ for every $\sigma_o \in \mathcal{C}_O(\mathcal{D},t)$.

3.1.4 4th step : Align unifiers according to value for global unifier

This step consists in splitting the above inclusion into a more parts according the value of the global variable; which induce to perform the second factorization using the global variable. This is achieved through the notions of *alignment unifier* and *aligned partition problem*. These notions are defined w.r.t. a given output unifier σ_O, a subset $\mathcal{C}$ of $\mathcal{C}_I(\mathcal{D},t)$ and a global variable x_G of $\mathcal{D}$. If $\gamma_\mathcal{C}$ denote the most general unifier of $\sigma_O(x_G)$ and all $\sigma(x_G)$ for $\sigma \in \mathcal{C}$, then the **aligned partition problem** for $\mathcal{C}$ is to partition symbolically all terms T in $Ground(\{\gamma_\mathcal{C}(\sigma(x_G))\}$ into two sets $[\mathcal{P}^+_{\sigma_\mathcal{O},\mathcal{C}}, \mathcal{P}^-_{\sigma_\mathcal{O},\mathcal{C}}]$[2] according to the condition.

$$\text{Ground}(\sigma_O\gamma_\mathcal{C})|_{[x_G \longrightarrow T]} \subseteq \text{Ground}(\{\sigma\gamma_\mathcal{C} \mid \sigma \in \mathcal{C}\})|_{[x_G \longrightarrow T]} \quad (*)$$

The unifier $\gamma_\mathcal{C}$ is called the **alignment unifier** corresponding to $\mathcal{C}$. The relation between this new refined inclusion and the inclusion in the previous step is stated by proving the lemma 5 in [Sch96].

3.2 Expanding symbolic family of place sets

This operation consists in extending a given set of (interpretation) of symbolic family of place sets chosen to be siphon with a new symbolic family of place sets. The more logical choice of such new candidate family is to select an input place (with its input terms) related to a transition that puts tokens in the current siphon and none of its input place are in this siphon. In other words, select a family of place which violate the current siphon. More technical treatment about this choice is given in [Sch96].

[2]Where $\mathcal{P}^+_{\sigma_\mathcal{O},\mathcal{C}}$ denote the different substitution for the global variable which respect the inclusion (*) and $\mathcal{P}^-_{\sigma_\mathcal{O},\mathcal{C}}$ those that don't respect the inclusion and thereby excluded for the siphon.

Chapter 4

A deductive system for siphon computation

It is worth mentioning that our main objective is not to introduce a new technique for computing siphons, but rather to rehabilitate the above technique for deriving a more *declarative* formulation that can be directly implemented using rewriting techniques. Nevertheless, the proposed formulation by exploiting as much as possible the fact that the net behaviour is expressed using rewrite rules, we show that efficiency is also addressed compared to the above technique.

Following that, first, we present an appropriate data structure that we use for verifying as well as computing siphons. Second, we informally and gradually explain with the help of the above example the different steps we followed for such verification / computation as well as the associated general inference rules. Third, we present a short comparison of this method with the one in [Sch96].

4.1 A data structure for verifying / computing siphons

For motivating the need for introducing a suitable data structure as well as its contents for an effective but yet declarative computation / verification of siphons, let us point out some features which remain unspecified in the above technique. First, there is no indication about the relation between an already verified part of family of siphons and a newly introduced family of places as extension. Indeed, as we see hereafter, a clear distinction between the two parts allows for deriving an efficient and easily controlled verification. Second, the method does not mention how to achieve backtracking, in order to derive all possible list of siphons from an initial choice.

To explicitly coping with these aspects and thereby keep trace of all necessary information in computing / verifying siphons in an algebraic net, we propose a data structure—as a tuple— that we denote by `SpDas` those components may be motivated as follows. First, important is the list of current (interpretation of) families of place sets considered to be composing the current siphon— already checked or initially chosen. Second, relevant is also the list of transitions, in our cases just the rewrite rule labels, already used during the extension from an initially chosen siphon. This allows us in particular for exploring all

different possible extensions, from an initial siphon, using backtracking in the same way as in programming logic. The third relevant information is the current family of place sets used as extension and not yet completely checked against the already existing family of siphons. Such verification have of course to be achieved by traversing all transitions which put (resp. take) tokens in the siphons and take (resp. put) tokens from the current extension family. This conduct us to take into account as fourth component the list of already used transitions for such verification after each extension step.

All for all, the data structure we propose, subsequently referred to as `SpDas`, is a tuple of the form:

$\langle\ Set\_E \mid Set\_V \mid Set\_Sph \mid Set\_Candt\ \rangle$.
Where :

- *Set_E* : Stands for a set[1] of transition labels that have been already traversed for the verification or the expansion until now w.r.t. a given initial siphon.
- *Set_V* : it contains the set of transition labels already used for the verification of siphon property at a given verification phase (defined later).
- *Set_Sph* : it represents a set of (interpretation of) family of place sets considered to be the current siphon.
- *Set_Candt* : it describes those family of terms which should be included (as expansion) in the current siphon in the case of success of the check process.

4.2 Description of the steps to be followed for verifying / computing siphons

With respect to this data structure, we present in the following different steps that have to be processed for resulting in a siphon.

4.2.1 1st Step : Initialization of the structure

Like the method in [Sch96], we have to start with a given place with which we associate an arbitrary (global) variable. However, in contrast to this method, we propose to choose a place appearing in the *right hand side* of a given transition[2]. Let $P_{in}.X_g$ be this initial symbolic family of place set. As illustrated in figure 4.1, the fact that the variable X_g may be substituted by any term of sort P_{in} is graphically illustrated by a whole black circle; which will be reduced more and more during the check and the expansion processes.

The corresponding initial state of our SpDas data structure will be as follows:

$$\langle \emptyset \mid \emptyset \mid P_{in}.X_g \mid \emptyset \rangle$$

.

[1] a set and not a list, so there is no duplication.

[2] Such rule always exists under the assumption that there is no isolated place or transition without input and output places.

Example 4.2.1 *Assume we would like to search for all siphons including the place* $\overline{WRI}$. *In this case the initial state takes the form:* $\langle \emptyset \mid \emptyset \mid \overline{WRI}.X_g \mid \emptyset \rangle$.

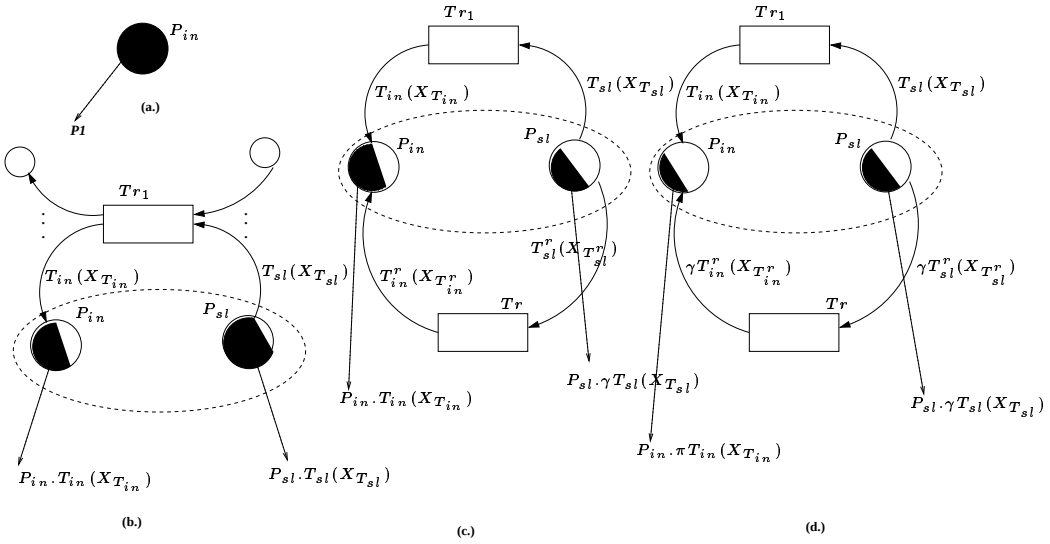

Figure 4.1: A simplified illustration of different steps to be followed for siphon computation-verification

4.2.2 Expansion of *SpDas* with a new candidate

As depicted in figure 4.1(b), after this initialization we have to select one rewrite rule which *puts* tokens in this initially chosen—as siphon— place . This means that such rule should have in the right hand side a subterm of the form $P_{in}.\surd$. From this rule, we have to select only *one input place* to be candidate for a future expansion of this siphon. Let the selected rewrite rule as follows:

$$Tr_1(X_{Tr_1}) : \langle P_1.T_1(X_{T_1}), .., \mathbf{P_{sl}}.\mathbf{T_{sl}}(\mathbf{X_{T_{sl}}}), .. \rangle \Rightarrow \langle Q_1.T_1'(X_{T_1'}), .., \mathbf{P_{in}}.\mathbf{T_{in}}(\mathbf{X_{T_{in}}}), .. \rangle$$

Where, $P_{sl}.T_{sl}(X_{T_{sl}})$ represents the selected (input) symbolic family of place sets.

According to the meaning of different components in the data structure *SpDas* and to the principle of expansion, the new resulted state from the above initial state takes the following forms :

$$\langle Tr_1(X_{Tr_1}) \mid \emptyset \mid \mathbf{P_{in}}.\mathbf{T_{in}}(\mathbf{X_{T_{in}}}) \mid \mathbf{P_{sl}}.\mathbf{T_{sl}}(\mathbf{X_{T_{sl}}}) \rangle$$

In other words, the new state of SpDas is composed of :

- The first component is the selected transition Tr_1 which puts tokens in the chosen siphon.
- Because no verification is yet performed, the second component empty.
- The third component is the selected family of siphons, where the free variable X_g is restricted to those terms appearing in corresponding right hand side of the selected transition (i.e. T_{in}). In the general case of expansion—not necessarily after the

initial state—, axiomatized hereafter, this variable binding should be regarded as a (an output) *unification* between terms attached with the siphon places (which is here just a variable) and the terms appearing in the corresponding (output) places of the chosen transition (T_{in} in this case). We denote such unification by σ_o.

- The last component is the first symbolic family of place sets, namely $P_{sl}.T_{sl}(X_{T_{sl}})$, to be selected as candidate in the siphon including the initial family $P_{in}.T_{in}(X_{in})$. This selected family of terms should also be regarded, in the general case, rather as an interpretation $P_{sl}.\sigma_o T_{sl}(X_{T_{sl}})$.

Improvement of the rule selection and SpDas

It is worth mentioning that for fulfilling the siphon properties either w.r.t. the expansion or the check process, what is always in common is that *only* one input place and one output place have to be selected in any processed transition. This is more explicit in the above expansion process, where except for the places P_{in} and P_{sl} and their associated terms in the selected transition (Tr), all the other (input / output) places of this transition play no role. Moreover, unless in case of backtracking[3], the chosen rewrite rule will never be re-selected again with the same initial siphon. From this observation come into light the ideas of *splitting* all the net rewrite rules in such a way that each derived rewrite rule contains exactly one input and one output place.

Of course, we should achieve such splitting in such a way that we keep trace of the label of the original transition; for this aim we use subscribers for the derived transitions. More precisely, for each transition modeled as a rule of the form:

$$Tr(X_{Tr}) : \langle P_1.T_1(X_{T_1}), ..., P_n.T_n(X_{T_n})\rangle \Rightarrow \langle Q_1.T'_1(X_{T'_1}), ..., Q_m.T'_m(X_{T'_m})\rangle$$

We propose to split it to the following $n \times m$ rewrite rules.

$$Tr(X_{Tr})(1) \ : \ \langle P_1.T_1(X_{T_1})\rangle \Rightarrow \langle Q_1.T'_1(X_{T'_1})\rangle$$

$$\ldots.$$

$$Tr(X_{Tr})(m) \ : \ \langle P_1.T_1(X_{T_1})\rangle \Rightarrow \langle Q_1.T'_m(X_{T'_m})\rangle$$

$$\ldots.$$

$$Tr(X_{Tr})(n*m) \ : \ \langle P_n.T_n(X_{T_n})\rangle \Rightarrow \langle Q_1.T'_m(X_{T'_m})\rangle$$

Note that we keep unchanged the label (to which we add a subscriber) of the transition. In other words, all transitions having the same label (and different subscribers) are considered as referring to the same transition. In all the rest of this paper we will assume this splitting process as given.

Example 4.2.2 *For our running example this splitting operation results in the following set of rewrite rules (derived directly from the rules in example 3.2.)*

[3]The backtracking consists in choosing another input place than P_{sl} and apply the subsequent reasoning for this candidate.

$\mathbf{ew(1)}$: $\langle WRI.x\rangle \Rightarrow \langle \overline{WRI}.x\rangle$
$\mathbf{ew(2)}$: $\langle WRI.x\rangle \Rightarrow \langle SEM.N\rangle$
$\mathbf{rq(1)}$: $\langle \overline{WRI}.x\rangle \Rightarrow \langle PND.x\rangle$
$\mathbf{rq(2)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle PND.x\rangle$
$\mathbf{bw(1)}$: $\langle PND.x\rangle \Rightarrow \langle WRI.x\rangle$
$\mathbf{bw(2)}$: $\langle SEM.N\rangle \Rightarrow \langle WRI.x\rangle$
$\mathbf{bw(3)}$: $\langle PND.x\rangle \Rightarrow \langle KEY.\bullet\rangle$
$\mathbf{bw(4)}$: $\langle SEM.N\rangle \Rightarrow \langle KEY.\bullet\rangle$

$\mathbf{er(1)}$: $\langle REA.y\rangle \Rightarrow \langle \overline{REA}.y\rangle$
$\mathbf{er(2)}$: $\langle REA.y\rangle \Rightarrow \langle SEM.y\rangle$
$\mathbf{br(1)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle REA.y\rangle$
$\mathbf{br(2)}$: $\langle SEM.y\rangle \Rightarrow \langle REA.y\rangle$
$\mathbf{br(3)}$: $\langle \overline{REA}.y\rangle \Rightarrow \langle REA.y\rangle$
$\mathbf{br(4)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle KEY.\bullet\rangle$
$\mathbf{br(5)}$: $\langle SEM.y\rangle \Rightarrow \langle KEY.\bullet\rangle$
$\mathbf{br(6)}$: $\langle \overline{REA}\rangle \Rightarrow \langle KEY.\bullet\rangle$

Remark 4.2.3 *According to this rule-splitting process, the above constructed* SpDas *is adapted to:*

$$\langle Tr_1(X_{Tr_1})(i) \mid \emptyset \mid P_{in}.T_{in}(X_{T_{in}}) \mid P_{sl}.T_{sl}(X_{T_{sl}})\rangle$$

Hence, we assume that the selected (split) rewrite rule results among other in: $\mathbf{Tr_1(X_{Tr_1})(i)} : \langle P_{sl}.T_{sl}(X_{T_{sl}})\rangle \Rightarrow \langle P_{in}.T_{in}(X_{T_{in}})\rangle$.

Example 4.2.4 *From the initial state,* $\langle \emptyset \mid \emptyset \mid \overline{WRI}.X_g \mid \emptyset\rangle$, *in the above example and according to the sketched expansion principle, the single rewrite rule to be selected is* `ew(1)` *that puts tokens in* $\overline{WRI}$. *The next state of SpDas then takes the form:*

$$\langle ew(1) \mid \emptyset \mid \overline{WRI}.X_g \mid WRI.X_g\rangle$$

Note that we have achieved a trivial unification of the variable X_g *with the variable* x.

Generalization of the expansion process

From this simplified illustration, we can derive the general form of expansion that we formulate as an inference rule, denoted as *Expand* and taking the form:

The Expand inference rule:

$$\frac{\langle Set\_E \mid Set\_V \mid Set\_Sph \mid Set\_Candt\rangle}{\langle Set\_E\ , Tr(k) \mid \emptyset \mid Set\_Sph\ , Set\_Candt \mid P_l.\sigma T^l\rangle}$$

$$if$$

$$Tr(k) : \langle P_l.T^l\rangle \Rightarrow \langle P_r.T^r\rangle \wedge Tr(\_) \notin Set\_E \wedge Tr(\_) \notin Set\_V \wedge P_r.T_I \in \{Set\_Sph \cup Set\_Candt\} \wedge P_l.\surd \notin \{Set\_Sph \cup Set\_Candt\} \wedge \sigma = MGU(T^r, T_I)$$

Under the condition that this 'extension' inference rule should be performed *only* after exploring all possibilities of verification (detailed hereafter), if the transition $Tr(k)$ does not exist then we can conclude that the set of family $\{Set\_Sph \cup Set\_Candt\}$ is a (set of) *siphon*.

Remark 4.2.5 *In this inference rule, T_I denotes any arbitrary terms associated with the place P_r. The conjunction $\wedge$ represents an 'and'. The condition $Tr(\_) \notin Set\_V$ means that for any subscriber (expressed by the underbar _) the rule with the identifier Tr must not have been used in the verification process. This is of course important because the extension should not concerns transitions which puts and takes tokens from the siphons. Notice also that in each application of this rule, the old set of candidates is shifted to the set of siphons and the set of verified rules in set to empty.*

4.2.3 The Check for the siphon property

Roughly speaking, in this crucial step we have to traverse all transitions which *put* (resp. *take*) tokens in (resp. from) the set of family already chosen or verified to be siphons (i.e. the initial as well as the already verified candidate set) and *take* (resp. *put*) tokens from the newly introduced candidate set, and verify that the definition of siphon holds.

As a first advantage of our splitting of the rewrite rules, is that, there is no more need for traversing **all** transitions to check if they put and/or take tokens from the selected siphon and the candidate. Indeed, if the set of siphons is $Set\_Sph = \{P_1.T_1, P_2.T_2, ..., P_k.T_k\}$ and the candidate is $Set\_Candt = \{P_c.T_c\}$ then we have to verify the siphon property directly for all rewrite rules of the form: $\mathbf{T_{ij}} : \langle P_i.T_i \rangle \Rightarrow \langle P_j.T_j \rangle$ and verifying the condition:

$$\text{if } P_i.\surd \in Set\_Sph \text{ then } P_j = P_c \text{ and vise versa}^4. \quad (**)$$

Before deriving the general inference rule for verifying the siphon property, let's explain using our simplified illustration as depicted in figure 4.1 the different elementary steps we follow for such verification.

Step 1. :

Select a rewrite rule that verify the above condition and which have not already been used for the verification (i.e. does not belong to the set Set_V). Let us denote by $Tr(j)$ such rewrite rule i.e. a rewrite rule reflecting a transition whose input and output arcs are in $\{P_{in}, P_{sl}\}$.

$$Tr(X_{Tr})(j) : \langle P_{sl}.T^r_{sl}(X_{T^r_{sl}}) \rangle \Rightarrow \langle P_{in}.T^r_{in}(X_{T^r_{in}}) \rangle$$

Step 2. :

Compute the (input most general) unifier between the term in the left hand side of this selected rule and the corresponding term (associated with the same place) in the siphon or the candidate set. In our simplified illustration, we have first to unify $T_{sl}(X_{sl})$ (i.e. the term selected from the siphon) with $T^r_{sl}(X_{T^r_{sl}})$ (i.e. the term appearing in the left hand side of the rule). Let $\gamma(X_\gamma)$ be most greater unifier (MGU) of such unification.

Step 3. :

Instantiation of the rewrite rule in consequence. The fact that we have achieved a unification between the *left hand side* of the selected rewrite rule and the corresponding term

[4] i.e. if $P_j.\surd \in Set\_Sph$ then $P_i = P_c$.

in the siphon or the candidate systematically imply that we have made an *instantiation* to all common variables between the right and the left hand side; henceforth it is necessary to take into account this instantiation in any next step. This instantiated rule is then as follows:

$$\mathbf{Tr}(\mathbf{X_{Tr}})(\mathbf{j}) : \langle P_{sl}.\gamma T^r_{sl}(X_{T^r_{sl}})\rangle \Rightarrow \langle P_{in}.\gamma T^r_{in}(X_{T^r_{in}})\rangle$$

Step 4. :

All what remains is to check if this transition that have taken tokens from the place P_{sl} it puts really tokens in the P_{in}. In other words, we have to compute the (subset of output) unifier between $\gamma T^r_{in}(X_{T^r_{in}})$—and not between $T^r_{in}(X_{T^r_{in}})$ and $T_{in}(X_{T_{in}})$. Let's denote by π such unifier.

By assuming that γ and π are non empty (see the remark below), we result in the following *SpDas* state, where both corresponding candidate and siphon terms have been instantiated in consequence and the rule (label) $Tr_1(X_{Tr_1})(j)$ has been added to the set of verified rules.

$$\langle Tr_1(X_{Tr_1})(i) \mid Tr_1(X_{Tr_1})(j) \mid P_{in}.\pi\ T_{in}(X_{T_{in}}) \mid\ P_{sl}.\gamma T_{sl}(X_{T_{sl}})\rangle$$

Remark 4.2.6 *Two particular cases should be taken into account in this verification process. The first case is when there is no (input) unifier γ. In this case if there is also no output unifier π the state of SpDas remains unchanged—because such transition neither takes nor puts tokens in the siphon and the candidate—, but if π is not empty we should compute its complement. However, another more simple way is to save this new state as $P_{in}.[\pi]^c\, T_{in}(X_{T_{in}})$; with the meaning of $[\pi]^c$ that for further verification all instantiations which coincide with π should not be considered. The third case is when π is empty which means that the transition takes tokens and puts noting. This case of course should be ignored, i.e. no change have to be processed in the corresponding SpDas.*

Generalization of the Check process

In the same spirit of the expansion process, we present the corresponding general inference rule that should applied for verifying the siphon property and this for all rewrite rule verifying the condition (**) described above. More precisely, under the condition that π and γ are non empty and the subscriber r and l may be exchanged (to capture the vice versa case), this inference rule takes the form.

The Check inference rule:

$$\frac{\langle Set\_E \mid Set\_V \mid Set\_Sph\ ,\ P_l.T_I \mid Set\_Candt\ ,\ P_r.T_J\rangle}{\langle Set\_E \mid Set\_V\ ,\ Tr(j) \mid Set\_Sph, P_l.\pi T_I \mid Set\_Candt, P_r.\gamma T_J\rangle}$$

$$if$$

$$Tr(j) : \langle P_l.T^l\rangle \Rightarrow \langle P_r.T^r\rangle \wedge Tr(\_) \notin Set\_V \wedge \gamma = MGU(T^l, T_I) \wedge \pi = MGU(\gamma T^r, T_J)$$

Remark that we have used the underbar '_' in $Tr(\_)$ to exclude more than one verification for rewrite rules with different subscribers but with same identifier label, i.e. representing the same transition (Tr).

Example 4.2.7 *In the previous example, by applying the expansion we have derived the following SpDas state.*

$$\langle ew(1) \mid \emptyset \mid \overline{WRI}.X_g \mid WRI.X_g \rangle$$

However, once again at this stage there is no possibility of verification (i.e. no rule whose both side are included in $\{\overline{WRI}.\surd, WRI.\surd\}$*. So we have to try again if there is possibility of expansion. The answer is yet with two possibilities: either using the rule* `bw(1)` *and adding the family* $PND.x$ *or the using the rule* `bw(2)` *and adding the family* $SEM.x$*. From the selection of the first rule (i.e.* `bw(1)`*) with the application of the inference rule of expansion we derive the following new state of SpDas (under a trivial substitution):*

$$\langle ew(1), bw(1) \mid \emptyset \mid \overline{WRI}.X_g, WRI.X_g \mid PND.X_g \rangle$$

Now we can check this siphon using the rewrite rule `rq(1)` *and the Check inference rule. However, due to different trivial substitution of* γ *and* π*, we result in the same state of SpDas up to the new introduced rewrite rule (in the verification list) i.e.*

$$\langle ew(1), bw(1) \mid rq(1) \mid \overline{WRI}.X_g, WRI.X_g \mid PND.X_g \rangle$$

At this stage, of course by considering the already processed transitions, namely `ew(1)`*,* `bw(1)`*,* `rq(1)` *as* `ew(_)`*,* `bw(_)`*,* `rq(_)`*, there neither possibility of using Check nor Expansion inference rules. This means that the set*

$$\{\overline{WRI}.X_g, WRI.X_g, PND.X_g\}$$

is a siphon where the (global) variable X_g *can be instantiated by either* D*,* E *or* F *leading to three set of siphons.*

Now using backtracking (formally approached hereafter), we can choose instead of the rule `bw(1)` *rather* `bw(2)`*, and follow the same reasoning. Another interesting case is the presence (or precisely the generation) of local variables. Let's start with the place* WRI *as a siphon. This imply that the initial state of our data structure is as follows:*

$$\langle \emptyset \mid \emptyset \mid WRI.X_g \mid \emptyset \rangle$$

For the expansion the rules `bw(1)` *and* `bw(2)` *are possible. By choosing the rule* `bw(2)` *and because* N *is equal to* $C \oplus D \oplus E \oplus$ *and referring to the way of our replacement of* $\oplus$ *with ',' in section 3, this rule should seen rather as three ones i.e. we have:*

$$\textbf{bw(2)} : \langle SEM.A \oplus SEM.B \oplus SEM.C \rangle \Rightarrow \langle WRI.x \rangle \iff \begin{array}{l} \textbf{bw(2)} : \langle SEM.A \rangle \Rightarrow \langle WRI.x \rangle \\ \textbf{bw(5)} : \langle SEM.B \rangle \Rightarrow \langle WRI.x \rangle \\ \textbf{bw(6)} : \langle SEM.C \rangle \Rightarrow \langle WRI.x \rangle \end{array}$$

So by choosing the (after this splitting) rule `bw(2)`*, we results in the following next* `SpDas` *state:*

$$\langle bw(2) \mid \emptyset \mid WRI.X_g \mid SEM.C \rangle$$

At this stage there is possible check with the rewrite rule `ew(2)`*, but for the same reason as* `bw(2)` *this rule corresponds to the following three rules:*

$\textbf{ew(2)} : \langle WRI.x \rangle \Rightarrow \langle SEM.A \rangle$

ew(3) : $\langle WRI.x \rangle \Rightarrow \langle SEM.B \rangle$

ew(4) : $\langle WRI.x \rangle \Rightarrow \langle SEM.C \rangle$

Of course we have to use the (after this splitting) rule `ew(2)` *for the verification, which results in the following next* `SpDas` *state:*

$$\langle bw(2) \mid ew(2) \mid WRI.X_g \mid SEM.A \rangle$$

At this stage there is also possibility of extension using the rule `er(2)` *through the unification of the variable y by A. The next state is then:*

$$\langle bw(2)\,,\, er(2) \mid \emptyset \mid WRI.X_g\,,\, SEM.A \mid REA.A \rangle$$

It is now easy to verify that there no further verification or expansion, which means that the set $\{WRI.X_g, SEM.A, REA.A\}$ *is a siphon where the variable* X_g *now play automatically the role of local variable. In other words, it has to be instantiated by D and E and F in the same set resulting in the siphon:*$\{WRI.D, WRI.E, WRI.F, SEM.A, REA.A\}$. *The same reasoning may be applied by choosing* `bw(5)` *and* `ew(3)` *or* `bw(6)` *and* `ew(4)`.

4.3 On the backtracking for deriving all siphons families

So far we have reported on how to derive a set of siphons starting from a given family of places, mainly by performing the two inference rules alternately until no more extension is possible—resulting in a set of siphons. subsequently, we refer to the last extension as final state of $SpDas$. In order to provide a systematic way to generate *all* siphon sets starting from a given family of places, mainly we have to adapt the extension inference rule so that it deals with the *backtracking*. The subscriber that we have used for distinguishing different rewrite rules associated with the same transition is very helpful here. Indeed, by assuming that in each extension we always choose a rule with a minimal subscriber, let's say $Tr(i)$, a simple backtracking—after reaching a final state with $Tr(i)$ as the last extended rule — is to attempt a new extension with the rule with same identifier and same output place but with a next subscriber, i.e. $Tr(i+1)$, and so on until the greater subscriber is reached. This process may be easily expressed as an inference rule:

[Case when next subscriber for the last rule in the first component exists]

$$\frac{\langle Set\_E\,,\, Tr(i) \mid Set\_V \mid Set\_Sph \mid Set\_Candt \rangle}{\langle Set\_R\,,\, Tr(i{+}1) \mid \emptyset \mid Set\_Sph \mid P_l.\sigma T^l \rangle\ [S, \{Set\_Sph, Set\_Candt\}]}$$

$$if$$

$$Tr(i) : \langle \surd . \surd \rangle \Rightarrow \langle P_r . \surd \rangle \wedge Tr(i+1) : \langle P_l . T^l \rangle \Rightarrow \langle P_r . T^r \rangle \wedge Tr(i+1) \notin Set\_E \wedge Tr(\_) \notin Set\_V \wedge P_r.T_I \in \{Set\_Sph\} \wedge P_l.\surd \notin \{Set\_Sph\} \wedge \sigma = MGU(T^r, T_I)$$

We have added the component $[S, \{Set\_Sph, Set\_Candt\}]$ just to indicate that the already computed family of siphons S—that we assume initially empty—have to be enriched with this new set of siphons, namely $\{Set\_Sph, Set\_Candt\}$.

However the most problem that remains is how to keep trace in a similar incremental way of rewrite rules with *different* identifiers used in a given extension and thereby complement this simple backtracking when all rules with same identifier and different subscribers have been explored. To cope with this problem, the best way[5] is to propose yet another structuring for the labels or the rule identifiers. More precisely, we propose to use a same identifier for all rewrite rules, for instance R, and besides the used subscriber for characterizing different rules associated with same transition, we use another subscriber for distinguishing different transitions. Thus, all our rewrite rules should be labeled by $R(i,j)$ where $R(i,\_)$ refers to the ith transition, whereas the j refers to its jth splitting.

Example 4.3.1 *With this convention the set of rewrite rules in example 4.2.2 is now as follows:*

$\mathbf{R(1,1)}$: $\langle WRI.x\rangle \Rightarrow \langle \overline{WRI}.x\rangle$

$\mathbf{R(1,2)}$: $\langle WRI.x\rangle \Rightarrow \langle SEM.N\rangle$

$\mathbf{R(2,1)}$: $\langle \overline{WRI}.x\rangle \Rightarrow \langle PND.x\rangle$

$\mathbf{R(2,2)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle PND.x\rangle$

$\mathbf{R(3,1)}$: $\langle PND.x\rangle \Rightarrow \langle WRI.x\rangle$

$\mathbf{R(3,2)}$: $\langle SEM.N\rangle \Rightarrow \langle WRI.x\rangle$

$\mathbf{R(3,3)}$: $\langle PND.x\rangle \Rightarrow \langle KEY.\bullet\rangle$

$\mathbf{R(3,4)}$: $\langle SEM.N\rangle \Rightarrow \langle KEY.\bullet\rangle$

$\mathbf{R(4,1)}$: $\langle REA.y\rangle \Rightarrow \langle \overline{REA}.y\rangle$

$\mathbf{R(4,2)}$: $\langle REA.y\rangle \Rightarrow \langle SEM.y\rangle$

$\mathbf{R(5,1)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle REA.y\rangle$

$\mathbf{R(5,2)}$: $\langle SEM.y\rangle \Rightarrow \langle REA.y\rangle$

$\mathbf{R(5,3)}$: $\langle \overline{REA}.y\rangle \Rightarrow \langle REA.y\rangle$

$\mathbf{R(5,4)}$: $\langle KEY.\bullet\rangle \Rightarrow \langle KEY.\bullet\rangle$

$\mathbf{R(5,5)}$: $\langle SEM.y\rangle \Rightarrow \langle KEY.\bullet\rangle$

$\mathbf{R(5,6)}$: $\langle \overline{REA}\rangle \Rightarrow \langle KEY.\bullet\rangle$

With this new label description, by choosing at each extension step always a rule with *minimal* numbers for both subscribers, the final state of *SpDas* may be abstracted—by illustrating just the first component *Set_E*—as $\langle R(i_1,j_1)\ ,\ \ldots\ ,\ R(i_k,j_k) \mid \ldots\rangle$. With that it became clear, first, we have to explore all extensions with $R(i_k, j_k+1)$,...,$R(i_k, j_{max})$. Second, we have to attempt all extensions with $R(nxt(i_k), j_{min})$, $R(nxt(i_k), j_{min}+1)$,...,$R(nxt(i_k), j_{max})$; where nxt is natural function that returns the first number greater than i_k such that the rule $R(nxt(i_k), j_{min})$ verify the extension conditions. Repeat such process with $nxt(nxt(i_k)), \ldots$ until no further rule exist. In this case we have to go one step back, i.e. using $R(i_{k-1}, j_{k-1})$ and apply the process repeatedly. In this way, it is easy to show that all extension possibilities may be explored. More precisely, by reaching a final state using the extension inference rule `Expand` (with the verification) one of the three following backtracking inference rules have to be applied exactly *one time* and then apply as usual alternatively the verification (at first), extension inference rules until a new final state is attained.

[Case when successor for the second subscriber of the last rule exist][6]

[5] Other tedious alternatives is to use another components that keep trace of these rules.

[6] This case is reformulation of the above case but using the new form of label.

$$\frac{\langle Set\_E\ ,\ R(i,k) \mid Set\_V \mid Set\_Sph \mid Set\_Candt \rangle}{\langle Set\_E\ ,\ R(i,k{+}1) \mid \emptyset \mid Set\_Sph \mid P_l.\sigma T^l \rangle\ [S,\{Set\_Sph,Set\_Candt\}]}$$

$$if$$

$$R(i,k+1) : \langle P_l.T^l \rangle \Rightarrow \langle P_r.T^r \rangle \wedge R(i,k+1) \notin Set\_E \wedge R(i,\_) \notin Set\_V \wedge P_r.T_I \in \{Set\_Sph\} \wedge P_l.\surd \notin \{Set\_Sph\} \wedge \sigma = MGU(T^r, T_I)$$

Note that at the beginning the set of expansion is initially empty.

[Case when successor for second subscriber of the last rule does not exist but next transition (i.e. next first subscriber) exist]

$$\frac{\langle Set\_E\ ,\ R(i,max) \mid Set\_V \mid Set\_Sph \mid Set\_Candt \rangle}{\langle Set\_R\ ,\ R(nxt(i),min) \mid \emptyset \mid Set\_Sph \mid P_l.\sigma T^l \mid\rangle\ [S,\{Set\_Sph,Set\_Candt\}]}$$

$$if$$

$$R(nxt(i),min) : \langle P_l.T^l \rangle \Rightarrow \langle P_r.T^r \rangle \wedge R(nxt(i),min) \notin Set\_R \wedge R(nxt(i),\_) \notin Set\_Ve \wedge P_r.T_I \in \{Set\_Sph\} \wedge P_l.\surd \notin \{Set\_Sph\} \wedge \sigma = MGU(T^r, T_I)$$

[Case when all rules with all subscribers for the last rule have been explored]

$$\frac{\langle Set\_E\ ,\ R(i,k)\ ,\ R(max_1,max_2) \mid Set\_V \mid Set\_Sph \mid Set\_Candt \rangle}{\langle Set\_E\ ,\ Tr(i,k{+}1) \mid \emptyset \mid Set\_Sph \mid P_l.\sigma T^l \mid \emptyset\rangle\ [S\ ,\ \{Set\_Sph,Set\_Candt\}]}$$

$$if$$

$$R(i,k+1) : \langle P_l.T^l \rangle \Rightarrow \langle P_r.T^r \rangle \wedge R(i,k+1) \notin Set\_E \wedge R(i,\_) \notin Set\_V \wedge P_r.T_I \in \{Set\_Sph\} \wedge P_l.\surd \notin \{Set\_Sph\} \wedge \sigma = MGU(T^r, T_I)$$

4.4 A comparison of our method with the one proposed in [Sch96]

As we have already pointed out, according to the references [Sch96, Sch97] there is no precise description on how the verification is achieved with respect a newly introduced (interpretation of) family of symbolic places for extending the already checked part of set of siphons. So, if we refer directly to the verification recalled in section 4 it is achieved by checking the *inclusion* of the ouptut unfier in the input unfier by exploring *all* transitions. In contrast to that, in our approach, first, only and exactly the *part*—i.e. as split rewrite rules verifying the condition indicated by (**)— of transitions that are directly concerned by such verification are explored. Second, because of this precise characterization of this set of rewrite rules that take and put tokens from the checked family of siphons and the candidate for extension, we have shown that it is more logical to first compute the input unfier (i.e. γ) and then instantiate the rule with respect to it and finally compute a subset of the output unifier. In other words, as we prove it hereafter, due to this precise characterization, instead of computing all output and all input unfier w.r.t. all transitions, we compute only the input unfier and *constrain* the output unifier to verify directly the inclusion by computing only the subset that verify the inclusion; and that only for those part of transitions that are really concerned by the verification.

The following result prove this enforcement of inclusion.

Lemma 4.4.1 *If the MGU π between $\gamma T^r_{in}(X_{T^r_{in}})$ and $T_{in}(X_{T_{in}})$ exists then there is a substitution σ such that $\sigma = \pi\gamma$.*

Proof 1 *The existence of π simply means that a unifer between $T^r_{in}(X_{T^r_{in}})$ and $T_{in}(X_{T_{in}})$, namely σ, exists even after binding, using γ, some variables of $X_{T^r_{in}}$ in $T^r_{in}(X_{T^r_{in}})$ by (sub)terms—to which should now correspond in $T_{in}(X_{T_{in}})$ either a variable or unifiable subterms. Henceforth, in the case when such variables are free it is more easier to find such unifer.*

As straightforwards result of this lemmas is what we call the **inclusion enforcement**.

Proposition 4.4.2 inclusion enforcement. *Under the above consideration and notations, the inclusion $\sigma_{X_g \cup X_{T^r}}$[7] $\subseteq \gamma_{X_g \cup X_{T^r}}$ always* **holds** *under the condition $\gamma \neq \emptyset$.*

Proof 2 *direct from the above lemmas and the definition of $\sigma_{X_g \cup X_{T^r}}$ that is equal to $\pi_{X_g \cup X_{T^r}} \gamma_{X_g \cup X_{T^r}}$*

Finally, note that our method using the proposed backtracking inference rules allows for systematically derive all possible sets of siphons.

[7] We abstract away from local variables in both substitutions.

Chapter 5

Conclusions

In this paper we addressed the problem of analysis of algebraic Petri nets by interpreting them in true concurrency semantics, namely rewriting logic. More specifically, we have rehabilitated the technique of verification / computation of siphons (and traps) proposed by K.Schmidt in [Sch96]. This rehabilitation has been achieved, first, through an adequate interpretation of net behaviour in rewriting logic using an appropriate form of rewrite rules. Second, for both check and expansion procedures, we proposed two easy-understandable inference rewrite rules. Third, we showed that with the help of rewrite rules, instead of computing separately all input and output unifier, it suffice to force this inclusion. Fourth, by making a clear distinction between already checked set of siphons and new candidate for its extension, we demonstrated how only those part of transitions really concerned by the verification have to be explored. Moreover, by proposing a suitable way of represented rewrite labels we derived simple inference rules for exploring all possibilities in generating siphons.

As near future work, we plan to implement these inference rules in the Maude language and perform it with more complex examples. This would allow us particularly to assess the limit as well as the advantages of the method for real applications. Another direction which we are curretly investigating is to what extend this method can be adapted and/or extended to take advantage of the ideas on liveness for CPNets forwarded in [BDH93]. On the other hand, and in order to consolidate the thesis that rewrite logic can be very helpful for deriving more effective and efficient analysis methods from existing ones, we plan investigate also techniques used for computing place and transition invariants.

Bibliography

[BDH93] K. Barkaoui, C. Dutheillet, and S. Haddad. An Efficient Algorithm for Finding Structural Deadlocks in Colored Petri Nets. In *Proc. of 14th Int. Conf. on Application and Theory of Petri nets*, Lecture Notes in Computer Science, Springer-Verlag, 1993.

[BGCM94] E. Battiston, V. Grespi, F. D. Cindio, and G. Mauri. Semantics Frameworks for a Class of Modular Algebraic Nets. In M. Nivat, C. Rattray, T. Russ, G. Scollo, editor, *Proc. of 3th. International AMAST Conference*, 1994.

[BM92] M. Bettaz and M. Maouche. How to Specify Non Determinism and True Concurrency with Algebraic Term Nets. In M. Bidoit, and C. Choppy, editor, *Proc. of 8th Workshop on Abstract Data Types*, Lecture Notes in Computer Science, Vol. 655, pages 164–180, 1992.

[BMSB93] M. Bettaz, M. Maouche, M. Soualmi, and S. Boukebeche. Protocol Specification using ECATNets. *Reséaux et Informatique Répartie*, 3(1):7–35, 1993.

[CDE+99] M. Clavel, F. Duran, S. Eker, J. Meseguer, and M. Stehr. Maude : Specification and Programming in Rewriting Logic. Technical report, SRI, Computer Science Laboratory, March 1999. URL : http://maude.csl.sri.com.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320, Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, 1990.

[EM85] H. Ehrig and B. Mahr. Fundamentals of algebraic specifications 1 : Equations and initial semantics. *EATCS Monographs on Theoretical Computer Science*, 21, 1985.

[GKM87] J. Goguen, C. Kirchner, and M. Meseguer. Concurrent Term Rewriting as a Model of Computation. In Keller, R. and Fasel, J., editor, *Proc. Workshop on Graph Reduction, Santa Fe, New Mexico*, Lectures Notes in Computer Science, Vol. 279, pages 53–93, 1987.

[GWM+92] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ. Technical Report No. SRI-CSL-92-03, Computer Science Laboratory, SRI International, 1992.

[JR91] K. Jensen and G. Rozenberg. *High-level Petri Nets.* Springer-Verlag, 1991.

[Mes90] J. Meseguer. Rewriting Logic as a unified Model of Concurrency. In Baeten, J. C. M. and Klop, J. W., editor, *13th Proc. of CONCUR'90*, Lectures Notes in Computer Science, Vol. 458, pages 384–400, 1990.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model for concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[Mes98] J. Meseguer. Research Directions in Rewriting Logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany*, Springer-Verlag, 1998.

[Rei91] W. Reisig. Petri Nets and Abstract Data Types. *Theoretical Computer Science*, 80:1–30, 1991.

[Sch96] K. Schmidt. How to Calculate Symbolically Siphons and Traps for some Algebraic Petri nets. Technical report, Helsinki University of Technology, Otaniemi, 1996.

[Sch97] K. Schmidt. Verification of Siphons and Traps for algebraic Petri nets. In P. Azema and G. Balbo, editors, *Proc. of 18th Int. Conf. on Application and Theory of Petri nets*, Lecture Notes in Computer Science, pages 427–446, Springer-Verlag, 1997.

[Ste98] M.-O. Stehr. A Rewriting Semantics for Algebraic Petri Nets. Manuscript, SRI International, March 1998.