ACCESS[1]
AlgebraiC Concurrent Events for System Specification

Didier Buchs
LGL-DI, Software Engineering Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne, Switzerland
buchs@di.epfl.ch

Giovanna Di Marzo
CUI, Centre Universitaire d'Informatique
University of Geneva
CH-1211 Geneva 4, Switzerland
dimarzo@cui.unige.ch

[1]This work is partly supported by Swiss National Fund for Scientific Research (FNSRS) grant FNRS 21.32286.91

Abstract

ACCESS is a new algebraic specification formalism, which focuses on the fine description of the "true" concurrency and on a high degree of expressivity.

Systems are specified by a set of local states, whose value changes under the occurrence of events. Both events and data structure are specified by abstract data types. Static properties, i.e. global constraints over events and data structure, are described by first order formulae, while dynamic axioms, explaining the behavior of events, are given by causality rules.

Concurrency can be described in different ways, it can be interleaving or true concurrency. Finally, expressivity is given by fine descriptions of both static and dynamic properties.

ACCESS has been demonstrate to be a natural generalization of a great variety of Petri Nets (as for example -Algebraic, -Coloured, -With arc extensions Petri Nets). It is also able to capture concepts from other formalisms as Gamma language or CO-OPN.

This report presents a complete description of ACCESS syntax and semantics, together with an example of ACCESS specification based on Petri Nets with Arc Extensions. This report also explains how specifications written in other languages as Petri Nets, Gamma, CO-OPN, are written in ACCESS.

Contents

Chapter 1

Introduction

There exists a great variety of formal languages or formalisms for systems specification, each of them addressing one or more properties like concurrency, modularity, description of data structure, etc.

We can mention the wide family of Petri Nets, which were firstly intended to describe concurrency between processes. Some extensions, in different directions, have been provided to the basic Petri Nets. Petri Nets have been extended with coloured tokens [Jens92], instead of only black tokens, in order to handle Petri Nets of small sizes, even for wide systems. The whole description of data structure handled by the system is possible, when we use Petri Nets with algebraic specifications [Reis91]. Another recent extension made to Petri Net integrates the notions of modularity and hierarchy [Guel94]. Usual properties of Petri Nets test the minimal value a place must have before a transition is fired. Some more complex conditions have been introduced by [Jens94], by defining not only input and ouptut arcs between places and transitions, but also tests and inhibitors arcs for testing special conditions. As a formalism derived from Pretri Nets, we mention the CO-OPN language [Buchs92], which is heavily based on algebraic petri nets, integrating notions of object-based as well as synchronization between objects.

Another wide family of formal languages is given by the process algebra, as for example the ACP language [Baet90], specifying systems by their behaviour. Predefined operators are provided to produce new behaviours from previous ones, as for example operators for parallelism, sequentiality or non determinism. way. The parallelism defined by ACP corresponds to the interleaving, i.e. two events, a, b, occur in parallel if there is the following choice: a occurs first and then b occurs, or b occurs first and then a.

While Petri Nets and Process Algebra describe, what we can call "computation" details of how exactly occurs the parallelism, the Gamma language [Ban93] is attached to the description of a logical parallelism, with no matter of a further implementation. A Gamma program consists in a collection of reactions able to occur in any order and at any time, in that way attention focuses only on possible events occuring in the system. This language is particularly well adapted for specification of reactive systems.

In this short list of examples of formal languages, we observe that most of them are based on algebraic specifications, extremely useful to describe data structure handled by systems. Concurrency is present in various forms as the interleaving with predefined operators for algebra of processes, or parallelism given by Petri Nets, or logical parallelism of Gamma programs. Moreover, modularity and object-oriented features are also controlled by most languages.

The formalism presented here, Access, tends to be a formalism large enough to capture features of other languages, in order to offer a generalization of other formal languages. One of the motivations to have a generalization language, is that a common formalism is useful for the study of properties of the captured languages, as well as the comparison between generalized languages, e.g. see if models for Petri Net specifications are equivalent to models for other specification languages.

Another crucial point, addressed by Access, concerns the description of concurrency. Concurrency is generally predefined, Access allows user-defined concurrency, so that in a given system you can have interleaving, or concurrency of Petri Nets, or some other "true" concurrency, or even a mixing of these previous concurrencies.

Following the logic of a generalization language, we have provided ACCESS with first order formulae allowing, in that way, the specification of a great variety of more or less complex properties.

ACCESS stands for Algebraic Concurrent Events for System Specification, as a system is specified by the mean of algebraic specifications for data structures, and by events, changing the state of the system.

Events are described in a double way: firstly events are considered as a static data and thus are defined with algebraic specifications (as standard data structures), secondly events are seen as dynamic entities whose behaviour changes the state of the specified system. When events are seen as data, they can be compound with operators in order to produce more complex events: we can define an operator called //, for example, that allows the handling of an event, which represents a possibly large set of other events occurring all in parallel. Events considered as data can also have their static properties be described by special axioms, called *Static Axioms*. When events are seen as dynamic entities changing the state of the system, their behaviour is captured by special axioms, *Dynamic Axioms*, allowing the description with all required details of their behaviour.

Section 2 presents an informal overview of ACCESS. The basic concepts underlying ACCESS are presented as well as examples of the way they are used to specify systems. Section 3 gives the whole syntax of ACCESS. This section consists in formally defining ACCESS presentations, which are the mean used in ACCESS to specify systems. Section 4 is dedicated to the semantics attached to ACCESS presentations, it gives the evaluation of all syntactical elements and defines which structures are considered as models. Section 5 focuses on examples of ACCESS presentations. It is entirely based on the case of Coloured Petri Nets (CPN) and the way how CPN specifications are transformed into ACCESS presentations. Section 6 gives the transformation into ACCESS presentations of algebraic petri nets, Gamma programs and CO-OPN specifications. Finally, section 7 ends this report with some concluding remarks about ACCESS characteristics.

Chapter 2

Overview of Access

ACCESS specifies systems by describing their data structures, and the changes of their state.

Data Structures are basically algebraic specifications. Changes of the system are defined by the mean of *events*, an event being responsible for a change occurring in a state of the system. Events actually change the global state of the system by modifying *local states*, i.e. parts of the global state. Operators working about events enable the writing of complex events. Events are defined by the constraints they have to check, and by their behavior, i.e. by the way they modify the state of the system.

This chapter informally presents the notions of local states, data structures and events. It then describes the Static Axioms, expressing constraints about sorts and operators; Dynamic axioms, describing the behavior of events; and Meta-rules, a mechanism to avoid the writing of large sets of Dynamic Axioms.

2.1 Local States

We want to express both the data structure handled by a system and the changes of states occurring in a system during its life. Changes of states are explained on the basis of Local States, that are smaller parts of the whole global state of a system. The set of Local States reflects the Global state of the system.

Example 2.1 *The system is subdivided in five local states: Lst_1, Lst_2, Lst_3, Lst_4, Lst_5. The global state of the system is given by the set of all local states. Each local state has a data sort and will take a value in an associated algebra. We see that Lst_1, Lst_2, Lst_5 are of the sort nat (for natural numbers), while Lst_3, Lst_4 are of the sort bool (for booleans). Each Local State has its own value, as 10 for Lst_1 or true for Lst_3.*

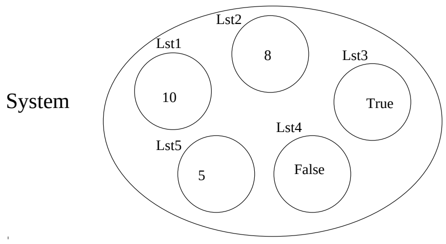

Figure 2.1: Local States of System

In ACCESS *this set of local states is defined as a special S-set, called V, where S is the set of available data sorts. In this example, we have:*

$$\begin{aligned} V_{nat} &= \{Lst_1, Lst_2, Lst_5\} \\ V_{bool} &= \{Lst_3, Lst_4\} \end{aligned}$$

2.2 Data Structures

Data Structures are described with usual Algebraic Specifications, i.e. data sorts, operations on sorts and conditions.

Example 2.2 *The algebraic specification of natural number and booleans is given by a set of sorts names, S and by a set F of operations over these sorts.*

$$\begin{array}{lllll} S & = \{nat, bool\} & & & \\ F & = \{ & zero: & & \rightarrow nat, \\ & & pred: & nat & \rightarrow nat, \\ & & succ: & nat & \rightarrow nat, \\ & & +: & nat\ nat & \rightarrow nat, \\ & & true: & & \rightarrow bool, \\ & & false: & & \rightarrow bool, \\ & & \neg: & bool & \rightarrow bool, \\ & & \vee: & bool & \rightarrow bool, \\ & & \wedge: & bool & \rightarrow bool\} \end{array}$$

2.3 Events

Events are also described by Algebraic Specifications. We then have event sorts, operations on events and the possibility to write conditions over events.

Moreover, it is possible to write any combination of events and to describe the details of the behavior of this combination.

Example 2.3 *The system of example 2.1 changes its state under the effect of event ev_1, or under event ev_2, or under the compound event $//(ev_1, ev_2)$. Event ev_1 changes the local states Lst_1 and Lst_2 to the new values 11 for the first one and 7 for the second one. Event ev_2 changes only Lst_3 to the value false. The compound event $//(ev_1, ev_2)$ changes the three local states simultaneously.*

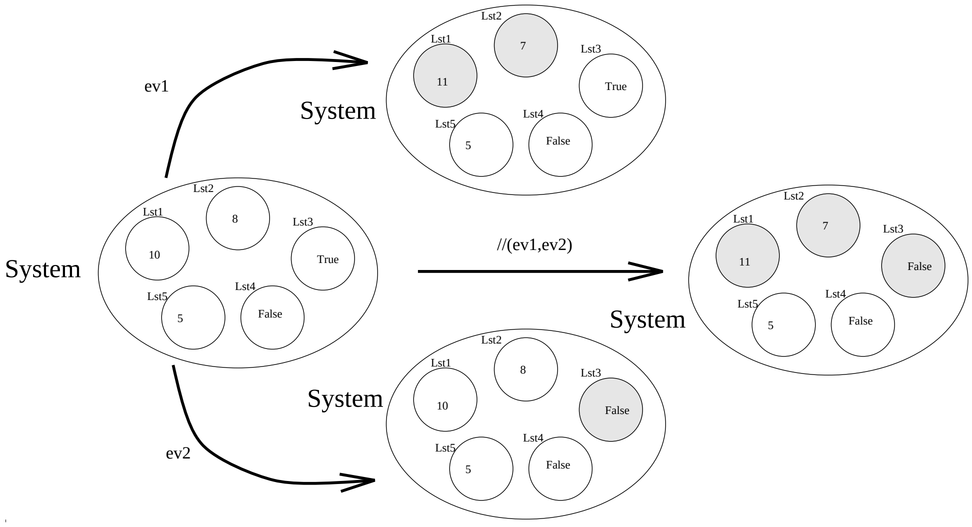

Figure 2.2: Changes of States under events

In ACCESS the event sorts are defined in a special set, called EV. Operations over events are part of the set F, as it is the case for operations over data. The event of our example are all of the same sort

ev, and each ev_i is an operation over the events, that returns an event of type ev. Similarly, $//$ is an operation over the events that takes two events of type ev and returns a new event of type ev.

$$
\begin{array}{lll}
EV & = \{ev\} & \\
F & = \{\ ev_1 : & \rightarrow ev \\
 & \quad\ ev_2 : & \rightarrow ev \\
 & \quad\ // : \quad ev\ ev & \rightarrow ev\}
\end{array}
$$

Remark 2.4 *The set of operations F may also contain operations over data structures and events simultaneously.*

2.4 Constraints: Static Axioms

Static Axioms are used to express static constraints of the system, they work on either data structures or events, e.g. the $+$ operator must have 0 as neutral element, or the $//$ operator must be commutative.

Static axioms are first order formula: *equational atoms*, *predicates*, or a combination of these two with the logical connectors $\neg, \exists, \forall, \Rightarrow, \vee, \wedge$.

Example 2.5 *The examples 2.2 and 2.3 give data sorts, event sorts and operations over these sorts, we give now some constraints that we can write in* ACCESS.

$$
\begin{array}{ll}
Ax & = \{\ \ \neg(D(Pred(zero)), pred(succ(n)) = n, n + zero = n, \ldots \\
 & \qquad \neg(True) = False, True \wedge a = a, \ldots \\
 & \qquad //(ev_1, ev_2) = //(ev_2, ev_1), \ldots\}
\end{array}
$$

Here $\neg(D(Pred(zero))$ is the predicate $D(Pred(zero)$ combined with the connector $\neg$; $pred(succ(n)) = n$ is an equational atom that constraints the $pred$ operator to be the inverse of $succ$ operator.

2.5 Description of Events: Dynamic Axioms

Static axioms about events, explain some constraints an event has to respect when it is seen as a data structure, but static axioms don't explain how exactly the event behaves to change the state of the system. To do that we have introduced: *Dynamic Axioms*, that explain how the state of the system is changed when an event occurs. Dynamic Axioms are made of *State Modification Formula* (changing the value a local state) and of *Static Axioms*.

Example 2.6 *Example 2.3 shows that when ev_1 occurs the value of the local state Lst_1 is increased by one, while the value of the local state ev_2 is decreased by one. The event ev_2 inverses the value of the local state Lst_3. Dynamic axioms describing these behaviors are:*

$$
\begin{array}{lll}
da_1 & = ev_1 : & Lst_1 := succ(Lst_1) \rightarrow \\
 & & \neg(Lst_2 = zero)\ \&\ (Lst_2 := pred(Lst_2)) \\
da_2 & = ev_2 : & \epsilon \rightarrow Lst_3 := \neg(Lst_3)
\end{array}
$$

The first dynamic axioms explains that ev_1 firstly changes the local state Lst_1 by its successor, it then checks if the local state Lst_2 is different from 0 and then changes this local state by its predecessor.

The second dynamic axiom describes ev_2, this event simply changes the value of local state Lst_3 to its inverse value.

For the case of the event $//(ev_1, ev_2)$ a dynamic axiom could be:

$$
\begin{array}{ll}
da_3 = //(ev_1, ev_2) : & Lst_1 := succ(Lst_1)\ \&\ \epsilon \rightarrow \\
 & \neg(Lst_2 = zero)\ \&\ (Lst_2 := pred(Lst_2))\ \& \\
 & (Lst_3 := \neg(Lst_3))
\end{array}
$$

This dynamic axiom is nothing else than the composition of the two previous axioms, we see that $//(ev_1, ev_2)$ *firstly behaves like the first parts of* ev_1 *and* ev_2 *and then continues with the second parts of* ev_1 *and* ev_2*.*

The arrow $\rightarrow$, appearing in meta-rules, provides a temporal ordering of the precondition (before the arrow) w.r.t the postcondition (after the arrow).

2.6 Meta-Rules

It is often necessary to write a large number (perhaps infinite) of Dynamic Axioms. Example 2.6) gives a dynamic axiom concerning the event $//(ev_1, ev_2)$, but there can be also another dynamic axiom for the event $//(ev_1, //(ev_1, ev_2))$, and so on. We see that there are an infinite number of such dynamic axioms. The notion of Meta-Rules helps the writing of Dynamic Axioms. Meta-Rules are made of two parts: some premices and an action. The premices, as well as the action, are meta-dynamic axioms. The premices stand for dynamic axioms (not meta-), and the action explains how a new dynamic axiom is obtained when the premices are replaced with dynamic axioms. Premices give the structure of dynamic axioms used to derive new dynamic axioms.

Example 2.7 *The following meta-rule describes the behavior of two events occurring in parallel:*

$$\overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1 \; ; \; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \qquad \leadsto \qquad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2$$

where $\overline{ev}_i$, $\overline{f}_i$, $\overline{g}_i$ *are meta-variables, that stands for syntactical elements.*

This meta-rule means that when an event is obtained by the $//$ operator over two other events $\overline{ev}_1, \overline{ev}_2$, the resulting event begins like the first part of the first event followed by the first part of the second event and ends like the second part of the first event followed by the second part of the second event.

According to this meta-rule, we obtain for the two events ev_1, ev_2 in place of $\overline{ev}_1, \overline{ev}_2$, the dynamic axiom da_3 of example 2.6. But with this (unique) meta-rule we also obtain the dynamic axiom for $//(ev_1, //(ev_1, ev_2))$ by:

$$\begin{aligned} da_4 = //(ev_1, //(ev_1, ev_2)) : \quad & (Lst_1 := succ(Lst_1)) \;\&\; (Lst_1 := succ(Lst_1)) \& \epsilon \rightarrow \\ & \neg(Lst_2 = zero) \;\&\; (Lst_2 := pred(Lst_2) \& \neg(Lst_2 = zero) \;\& \\ & (Lst_2 := pred(Lst_2)) \;\&\; (Lst_3 := \neg(Lst_3)) \end{aligned}$$

This new dynamic axiom says that $//(ev_1, //(ev_1, ev_2))$ begins with the precondition part of ev_1 and continues with the precondition part of $//(ev_1, ev_2)$, it ends with the postcondition of ev_1 and with the postcondition of $//(ev_1, ev_2)$. The result is that the local state Lst_1 is incremented by 2, while Lst_2 is decremented by 2 and Lst_3 is inversed.

Chapter 3

Syntax of Access

A system is specified in Access by four sets of syntactical elements: the signature, the local states, the variables and the axioms.

The signature is composed of abstract data types (data sorts and operations on sorts), and of abstract events types (events sorts and operations on events).

A local state is a part of the whole system, its sort is one of the data sorts of the signature. The set of all local states gives the global state of the system.

The variables, useful for specifying constraints upon the system, are of different sorts: variables of data and variables of events.

Finally, the axioms let us explain the adequacy of a model to the system. There are 3 different kinds of axioms. The static axioms are conditions concerning static properties about the abstract data types and the events, while dynamic axioms are intended to explain the changes occurring to the system. A dynamic axiom describes the effect of an event upon the state of the system. Besides these two basic types of axioms, there is a third one, the meta-rules, that are used to avoid the writing of all the dynamic axioms.

A system is then specified using abstract data types and events, whose constraints are given by static and dynamic axioms.

This chapter describes the signature, the local states, the variables, the terms and the axioms writable in Access. It then gives the presentation of a system.

3.1 Signature

The signature is made of abstract data types and of abstract events types. Abstract data types refer to the data structure the system manipulate, while events are responsible for the system evolution.

Above the sets of data sorts and event sorts, the set of operations lets us construct data with some other data or events, or lets us construct events with other events or data.

Both data sorts and event sorts are order-sorted sets.

This section firstly presents basic definitions about order-sorted sets, and then gives the signature definition.

Definition 3.1 *A partial order relation over a set E is a binary relation R such that:*

$$R \subseteq (E \times E)$$

and R is reflexive, transitive and antisymmetric.

Definition 3.2 *Given R_1, R_2 two partial order relations over the sets E_1, E_2 respectively. The extension of these two orderings to the strings of equal length in $(E_1 \cup E_2)^*$, is the partial ordering R defined by:*

$$(v'_1 \dots v'_n, v_1 \dots v_n) \in R \textit{ iff } (v'_i, v_i) \in (R_1 \cup R_2)$$

Definition 3.3 *Given R, a relation verifying the antisymmetric property, then R^* is the transitive and reflexive closure of R.*

Notation 3.4 *We denote by $R = \{\}^*$, the partial order relation made of the reflexive relations only.*

Definition 3.5 *A signature is a triple $\Sigma = (S, EV, F)$ where:*

- *S is a partially ordered set of data sorts:*

  $S = \{s_1, s_2, s_3, \ldots\}$, $\subseteq_S \subseteq (S \times S)$, $\subseteq_S$ *a partial ordering*

- *EV is a partially ordered set of events sorts:*

  $EV = \{ev_1, ev_2, ev_3, \ldots\}$, $\subseteq_{EV} \subseteq (EV \times EV)$, $\subseteq_{EV}$ *a partial ordering*

- *F is a set of operations over the data and events:*

  $F = \{op : w \rightarrow v \mid w \in (S \cup EV)^*, v \in (S \cup EV), op \text{ is an operation identifier}\}$
  F satisfies the monotonicity condition:
  If $op : w_1 \rightarrow v_1, op : w_2 \rightarrow v_2 \in F$ *and* $(w_1, w_2) \in \subseteq_{(S \cup EV)}$ *then* $(v_1, v_2) \in (\subseteq_S \cup \subseteq_{EV})$

Where $\subseteq_{(S \cup EV)}$ is the extension of $\subseteq_S$ and $\subseteq_{EV}$ to the strings of equal length in $(S \cup EV)^*$.

The signature is made of a partially order-sorted set of data sorts, a partially order-sorted set of events sorts, and of a set of operations over data and events.

The operations of F, have as parameters sorts, w, a list of either data sorts or events sorts or a combination of both data and events sorts. These operations have as result sort, v, a unique name of data or event sort.

The monotonicity condition [Gogu89] over overloaded operations ensures the result sort of an overloading function to be a supersort of the result sort of the overloaded function.

When an operation overloads another operation by using supersorts as parameters sorts, then the result sort must also be a supersort.

Definition 3.6 *Given E a set of sorts, and $\subseteq_E$ a partial order relation, we say that v' is a subsort and v is a supersort if (v', v) belongs to $\subseteq_E$.*

Notation 3.7 *We denote by $op_{w,s}$ the operation $op : w \rightarrow v$ of F.*

Remark 3.8 *The use of partial order relations over data sorts ($\subseteq_S$) and event sorts ($\subseteq_{EV}$) eases the writing of operations of F and, further, the writing of constraints. The principle is that what is valid for a given sort is valid for all its included sorts.*

Definition 3.9 *[Wirs90] Given a set E, an E-set A is a family $\{A_e\}_{ev \in EV}$ of sets indexed by E.*

Definition 3.10 *Given a set E and two E-sets, A, B, the set of all E-morphisms is given by:*

$$\{f : A \rightarrow B \mid f \text{ a total function, s.t. } f(A_e) \subseteq B_e, \forall e \in E\}$$

Definition 3.11 *Given a set E and two E-sets, A, B, the set of all E-functions is given by:*

$$\{f : A \rightarrow B \mid f \text{ is a partial function, s.t. } f(A_e) \subseteq B_e, \forall e \in E\}$$

Example 3.12 *An example of signature is given by the natural numbers, the booleans, the queues of naturals and the queues of booleans together with events working upon these data sorts.*

$$
\begin{array}{lll}
\multicolumn{3}{l}{\Sigma = (S, EV, F) \ where:} \\
S & = \{nat, bool, queue\_nat, queue\_bool\} & \\
EV & = \{ev\_nat, ev\_bool, ev\} & \\
F & = \{ \begin{array}[t]{lll}
zero: & & \rightarrow nat, \\
pred: & nat & \rightarrow nat, \\
succ: & nat & \rightarrow nat, \\
+: & nat\ nat & \rightarrow nat, \\
true: & & \rightarrow bool, \\
false: & & \rightarrow bool, \\
\neg: & bool & \rightarrow bool, \\
\vee: & bool & \rightarrow bool, \\
\wedge: & bool & \rightarrow bool, \\
emptyqueue\_nat: & & \rightarrow queue\_nat, \\
add\_nat: & nat\ queue\_nat & \rightarrow queue\_nat, \\
remove\_nat: & queue\_nat & \rightarrow queue\_nat, \\
first\_nat: & queue\_nat & \rightarrow nat, \\
emptyqueue\_bool: & & \rightarrow queue\_bool, \\
add\_bool: & bool\ queue\_bool & \rightarrow queue\_bool, \\
remove\_bool: & queue\_bool & \rightarrow queue\_bool, \\
first\_bool: & queue\_bool & \rightarrow bool, \\
get\_nat: & nat & \rightarrow ev\_nat, \\
put\_nat: & nat & \rightarrow ev\_nat, \\
get\_bool: & bool & \rightarrow ev\_bool, \\
put\_bool: & bool & \rightarrow ev\_bool, \\
//: & ev\_nat\ ev\_nat & \rightarrow ev\_nat, \\
//: & ev\_bool\ ev\_bool & \rightarrow ev\_bool, \\
//: & ev\ ev & \rightarrow ev\}
\end{array} &
\end{array}
$$

We have $\subseteq_S = \{\}^*$, $\subseteq_{EV} = \{(ev\_nat, ev), (ev\_bool, ev)\}^*$

Data sorts of this signature are nat, $bool$, $queue\_nat$, $queue\_bool$, and events sorts are ev_nat, ev_bool, ev. The event sorts, ev_nat, ev_bool, denote events working about natural numbers and booleans respectively, while the sort ev is a more general event sort. There is no order on data sorts. The order on event sorts states that the ev sort is greater than the other two event sorts.

Operators are either usual operators over natural numbers, booleans and queues ($zero, \vee, \ldots$), or operators producing events from a data only (get_nat, get_bool) or operators producing events from events (//). In the case of events produced from a data only, we call these events, *labelled events* and the data is the *parameter* of such events. In the case of events produced from other events, they will be called *compound events*. For example with // the resulting compound event is an event produced when two other events occur in parallel. This operator furnishes also an example of overloading, as the same operation identifier, //, is used over different sorts: ev_nat, ev_bool or ev. The monotonicity condition is verified as the ev_nat, ev_bool sorts are two subsorts of the ev sort.

Remark 3.13 *The use of operations, working on both events and data, is particularly useful to make an event become a data structure or vice-versa.*

3.2 Local States

An entire system is subdivided into local states, each of them reflecting a part of the system. The set of all local states composes the global state. A local state has a value whose sort is one of the sorts of S.

Definition 3.14 *V is the S-set of all local states.*

The local states have each a name and a data sort.

Example 3.15 *For an example of two queues, one of natural numbers and the other one of booleans, the set of local states is made of one local state of sort queue_nat and another one of sort queue_bool.*

$$\begin{array}{l} V = V_{nat} \cup V_{bool} \cup V_{queue\_nat} \cup V_{queue\_bool} \text{ where} \\ \ V_{nat} = \emptyset \\ \ V_{bool} = \emptyset \\ \ V_{queue\_nat} = \{Nqueue\} \\ \ V_{queue\_bool} = \{Bqueue\} \end{array}$$

In this example, there are only two local states, $Nqueue$, $Bqueue$, whose data sorts are *queue_nat*, *queue_bool* respectively.

3.3 Variables

Variables are used to symbolize either data structures or events. Therefore there are different types of variables, the variables of data sorts and variables of events sorts.

Definition 3.16 *X is the $(S \cup EV)$-set of all variables:*

$$\begin{array}{l} X = X_S \cup X_{EV} \text{ where:} \\ \ X_S \text{ is the } S\text{-set of variables of data} \\ \ X_{EV} \text{ is the } EV\text{-set of variables of events} \end{array}$$

Example 3.17 *Examples of variables are:*

$$\begin{array}{l} (X_S)_{nat} = \{m\}, (X_S)_{bool} = \{a, b\} \\ (X_S)_{queue\_nat} = \{s\}, (X_S)_{queue\_bool} = \{r\} \\ (X_{EV})_{ev\_nat} = \{evnt_1, evnt_2\}, (X_{EV})_{ev\_bool} = \emptyset, (X_{EV})_{ev} = \emptyset \end{array}$$

3.4 Terms

The operations, over data and events, let us construct data or events from other data or events. The terms are all these syntactical constructions.

Definition 3.18 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of terms $T_\Sigma(X, V)$ is the least $(S \cup EV)$-set such that:*

$$\begin{array}{rll} 1. & (X_S)_s \cup V_s & \subseteq (T_\Sigma(X,V))_s, s \in S \\ 2. & (X_{EV})_{ev} & \subseteq (T_\Sigma(X,V))_{ev}, ev \in EV \\ 3. & \left.\begin{array}{ll} t_i \in T_\Sigma(X,V)_{v_i}, i \in \{1,\ldots,n\} & and \\ op \in F, op: v_1 \ldots v_n \to v & then \end{array}\right\} & \Rightarrow op(t_1,\ldots,t_n) \in (T_\Sigma(X,V))_v \\ 4. & (v, v') \in (\subseteq_S \cup \subseteq_{EV}) \Rightarrow (T_\Sigma(X,V)_v \subseteq T_\Sigma(X,V)v') & \end{array}$$

Terms are either variables or local states, or combination of terms with operations of F. The partial orderings, $\subseteq_S, \subseteq_{EV}$ induce all terms of a subsort to be also terms of a supersort.

Example 3.19 *For the example of queues of naturals and queues of booleans, some terms can be:*

$$\begin{array}{ll} add\_nat(m, Nqueue), add\_bool(a \vee b, Bqueue) & \in (T_\Sigma(X,V))_{s \in S} \\ get\_nat(m), put\_nat(m), //(get\_nat(m), put\_nat(m)) & \in (T_\Sigma(X,V))_{ev \in EV} \end{array}$$

See example 3.17 for the types of variables.

Terms can be of either data sort or event sort. The data sort terms $add\_nat(m, Nqueue)$ and $add\_bool(a \vee b, Bqueue)$ stand for adding the natural value m, or the boolean value $a \vee b$, on the queue $Nqueue$ or $Bqueue$ respectively. The event sort terms $get\_nat(m)$, $put\_nat(m)$ stand for two labelled events having as parameters a natural number m. The last event sort term $//(get\_nat(m), put\_nat(m))$ stands for the event being the result of the two events $get\_nat(m)$, $put\_nat(m)$ occurring in parallel.

3.5 Axioms

Axioms are the part of the system specification that constraints an algebra to stick to the system. These axioms are of three different natures. First, we have *Static Axioms*, giving static properties about either data structures or events. Then we have axioms specially dedicated to the description of events. These *Dynamic Axioms* explain how a given event acts upon the system, and changes its state. Finally, special Axioms, *Meta-Rules*, are a syntactic mean used to ease the writing of a large set (possibly infinite) of dynamic axioms.

3.5.1 Static Axioms

Static Axioms can be made of traditional*Equational Atoms* about data structures or *Predicates* about data structure. But, Static Axioms may also contain Equational Atoms and Predicates about events.

Equational atoms are equalities between terms of the same sort (data or event sort). Predicates are used to express definition constraints upon the terms. Static axioms are then given by combinations of Equational Atoms and Predicates with the logical connectors $\forall \exists \neg \vee \wedge \Rightarrow$.

Static axioms can be used in a global way or in a local way. A global static axiom must be valid for all possible states of the system, while a local axiom has to be valid for the current state only.

Definition 3.20 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of* Equational Atoms *is given by:*

$$EA_\Sigma(X, V) = \cup_{v \in (S \cup EV)} \{t_1 = t_2 | t_1, t_2 \in T_\Sigma(X, V)_v\}$$

Definition 3.21 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of* Predicates *is given by:*

$$Pred_\Sigma(X, V) = \{D(t) | t \in T_\Sigma(X, V)\}$$

Definition 3.22 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of* Static (global) Axioms *is the least set recursively defined by:*

$$\begin{array}{ll} EA_\Sigma(X, V) & \subseteq SA_\Sigma(X, V) \\ Pred_\Sigma(X, V) & \subseteq SA_\Sigma(X, V) \end{array}$$

$$\left.\begin{array}{c} sa, sa_1, sa_2 \in SA_\Sigma(X, V) \\ x \in (X_S \cup X_{EV}) \end{array}\right\} \Rightarrow \left\{\begin{array}{ll} sa_1 \vee sa_2 & \in SA_\Sigma(X, V) \\ sa_1 \wedge sa_2 & \in SA_\Sigma(X, V) \\ sa_1 \Rightarrow sa_2 & \in SA_\Sigma(X, V) \\ \neg sa & \in SA_\Sigma(X, V) \\ \exists x.(sa) & \in SA_\Sigma(X, V) \\ \forall x.(sa) & \in SA_\Sigma(X, V) \end{array}\right.$$

Static (global) axioms are first order formula made of equational atoms or predicates combined together with logical connectors.

The further definition of *Dynamic Axioms* (see definition 3.27) needs Static Axioms that do not contain nor Equational Atom nor Predicates about events. We will call Static Axioms appearing inside Dynamic Axioms, Static *local* Axioms. The above definition of Static Axioms is actually the definition of Static *global* Axioms. We give now the dual definition of the Static *local* Axioms. This definition is essentially the same as those of global axioms excepts that no Equational Atom and Predicates about events are present.

Definition 3.23 *Let* $\Sigma = (S, EV, F)$ *be a signature,* X *a set of variables,* V *a set of local states, the set of* Static (local) Axioms *is the least set recursively defined by:*

$$\begin{array}{ll} EA_\Sigma(X,V)\backslash(EA_\Sigma(X,V))_{ev\in EV} & \subseteq SLA_\Sigma(X,V) \\ Pred_\Sigma(X,V)\backslash(Pred_\Sigma(X,V))_{ev\in EV} & \subseteq SLA_\Sigma(X,V) \end{array}$$

$$\left.\begin{array}{c} sla, sla_1, sla_2 \in SLA_\Sigma(X,V) \\ x \in X_S \end{array}\right\} \Rightarrow \left\{\begin{array}{ll} sla_1 \vee sla_2 & \in SLA_\Sigma(X,V) \\ sla_1 \wedge sla_2 & \in SLA_\Sigma(X,V) \\ sla_1 \Rightarrow sla_2 & \in SLA_\Sigma(X,V) \\ \neg sa & \in SLA_\Sigma(X,V) \\ \exists x.(sla) & \in SLA_\Sigma(X,V) \\ \forall x.(sla) & \in SLA_\Sigma(X,V) \end{array}\right.$$

Example 3.24 *Examples of Static global Axioms for the queues of naturals and booleans, are given by:*

$$\begin{array}{l} \neg(D(pred(zero))) \\ remove\_nat(add\_nat(m,s)) = s \\ remove\_bool(add\_bool(a,r)) = r \\ //(evnt_1, evnt_2) = //(evnt_2, evnt_1) \end{array}$$

The Predicate: $\neg(D(pred(zero)))$ means that the predecessor of zero is not defined. The Equational Atoms of $queue\_nat, queue\_bool$: $remove\_nat(add\_nat(m,s)) = s$ and $remove\_bool(add\_bool(a,r)) = r$ mean that adding a value to a queue and then removing a value from that queue doesn't affect the queue. The Equational Atom of ev_nat: $//(evnt_1, evnt_2) = //(evnt_2, evnt_1)$ means that the operator $//$ over ev_nat sort is commutative: combining two events of natural in parallel doesn't depend on the order of the combination.

3.5.2 Dynamic Axioms

Dynamic Axioms explain *how* an event modifies the state of the system. They consists of conditions the system has to meet before, during or after the event takes place, and of description of the system's state. A dynamic axiom is made of three parts, the first one is the labelled event that is to be explained by the axiom. The second and the third part are a collection of Static local Axioms combined with changes of local states.

The changes of a local state are called *State Modification Atoms*; the mergence of State Modification Atoms and Static Local Axioms are called *State Modification Formula.*

Definition 3.25 *Let* $\Sigma = (S, EV, F)$ *be a signature,* X *a set of variables,* V *a set of local states, the set of* State Modification Atom *is given by:*

$$SMA_\Sigma(X,V) = \cup_{s\in S}\{v := t \mid v \in V_s, t \in (T_\Sigma(X,V))_s\}$$

A state modification atom (sma) is made of the symbol $:=$, whose left part is given by a local state and whose right part is made of a term. The sort of the local state and the term have to be equal. The sma, $v := t$ means that the local state v changes its value to the value of the term t.

Definition 3.26 *Let* $\Sigma = (S, EV, F)$ *be a signature,* X *a set of variables,* V *a set of local states, the set of* State Modification Formula *is the least set recursively defined by:*

$$\begin{array}{ll}
1. & \varepsilon \in SMF_\Sigma(X,V) \\
2. & SMA_\Sigma(X,V) \subseteq SMF_\Sigma(X,V) \\
3. & SLA_\Sigma(X,V) \subseteq SMF_\Sigma(X,V) \\
4. & \left.\begin{array}{r} f_1 \in (\{\varepsilon\} \cup SMA_\Sigma(X,V) \cup SLA_\Sigma(X,V)) \\ f_2 \in SMF_\Sigma(X,V) \end{array}\right\} \Rightarrow f_1 \& f_2 \in SMF_\Sigma(X,V) \\
5. & \left.\begin{array}{r} f \in SMF_\Sigma(X,V) \\ x \in X_S \end{array}\right\} \Rightarrow \exists x.(f) \in SMF_\Sigma(X,V)
\end{array}$$

A State Modification Formula (smf) is either empty, ε, that is to say, there is no action and the state is not modified, or a State Modification Atom or a Static Local Axiom or a combination of two of these elements by the special connector &. Finally, a State Modification Formula may be overalled by the $\exists$ quantifier. This is very useful to explicit which smf supports which variables.

The symbol & is a syntactic symbol which doesn't contain any notion of precedence. It only means that the elements, connected with the & are involved in the same smf.

Definition 3.27 *Let* $\Sigma = (S, EV, F)$ *be a signature,* X *a set of variables,* V *a set of local states, the set of* Dynamic Axioms *is given by:*

$$DA_\Sigma(X,V) = \{evnt : f \rightarrow g \,|\, evnt \in (T_\Sigma(X,V))_{ev \in EV}, f, g \in SMF_\Sigma(X,V)\}$$

Dynamic axioms are causality rules made of three parts: (1) an event term $evnt$, representing the labelled or compound event, whose behavior is explained by the dynamic axiom; (2) an smf, f, representing a precondition, (3) an smf, g, representing a postcondition. The arrow, $\rightarrow$, provides a local temporal ordering as the precondition must occur before the postcondition.

Example 3.28 *For the above example, Dynamic Axioms about events are:*

$$\begin{array}{rl}
get_nat(m): & \neg(Nqueue = emptyqueue_nat) \rightarrow \\
 & (m = first_nat(Nqueue))\& Nqueue := remove_nat(Nqueue), \\
put_nat(m): & \varepsilon \rightarrow Nqueue := add_nat(m, Nqueue)
\end{array}$$

See example 3.17 for the types of variables.

The expression $\neg(Nqueue = emptyqueue\_nat)$, is a Static Local Axiom, while $(m = first\_nat(Nqueue)$ is an Equational Atom. The expressions $Nqueue := remove\_nat(Nqueue)$, $Nqueue := add\_nat(m, Nqueue)$ are examples of State Modification Formula.

These two Dynamic Axioms describe events of natural, the first one explains how works the event $get\_nat(m)$ and the second one the event $put\_nat(m)$. According to these Dynamic Axioms, the event $get\_nat(m)$ removes an element from the Local State $Nqueue$, if this local state is not an empty queue, the element removed from the queue is then stored in the variable m. The event $put\_nat(m)$, add the value m on the local state $Nqueue$. This event can always occur as there is not preliminary condition to check.

3.5.3 Meta-Rules

Meta-rules are a mean to avoid the writing of an infinite number of dynamic axioms. The need of meta-rules follows from the use of operators on events. For example, the // operator on events, operates in the same way for all events. Without meta-rules, it becomes necessary to explicitly write all the dynamic axioms, there would be one for //(ev1,ev2), another one for //(ev1,//(ev1,ev2)), and so on.

A Meta-rule gives the skeleton of a dynamic axiom. The actual dynamic axioms are later derived from the meta-rules and from previous other dynamic axioms.

A Meta-rule is made of two parts, some *Premices*: zero, one or more meta-dynamic axioms and an *Action*: one meta-dynamic axiom. Premices stands for previous actual dynamic axioms, that are used to derived the new dynamic axiom. The Action stands for the dynamic axiom to derive. Meta-dynamic axioms are dynamic axioms where the syntactical elements appearing inside are not fixed: meta-dynamic axioms are only the skeleton giving a more or less precise form of the dynamic axiom they represent.

The use of meta-rules implies the use of meta-variables. These special variables are useful to get an abstraction of the syntactical elements appearing in a dynamic axiom but not yet fixed.

When all the dynamic axioms will be derived from a meta-rule, all the syntactical elements appearing in the meta-rule will remain unchanged, and all the meta-variables will become syntactical elements. The difference between a variable of $X_S \cup X_{EV}$ (appearing in terms) and a meta-variable is that the variable stands for a future value of sort or event, and the meta-variable stands for a syntactical element.

Definition 3.29 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of meta-variables of terms is the $S \cup EV$-set, $\overline{X}_T$ such that $\overline{X}_T \cap X = \emptyset$. The set of meta-variables of smf is the set $\overline{X}_{SMF}$, such that $\overline{X}_{SMF} \cap (X \cup \overline{X}_T) = \emptyset$.*

Example 3.30 *Some meta-variables of terms and smf:*

$$\begin{array}{ll} (\overline{X}_T)_{ev\_nat} & = \{\overline{ev}_1, \overline{ev}_2\} \\ (\overline{X}_T)_{ev\_bool} & = \{\overline{ev}_3, \overline{ev}_4\} \\ (\overline{X}_T)_{ev} & = \{\overline{ev}_5, \overline{ev}_6\} \\ (\overline{X}_{SMF}) & = \{\overline{f}_1, \overline{f}_2, \overline{g}_1, \overline{g}_2\} \end{array}$$

The meta-variables used here are of two kinds, the meta-variables that stands for *event terms*: $\overline{ev}_1, \overline{ev}_2, \overline{ev}_3, \overline{ev}_4, \overline{ev}_5, \overline{ev}_6$ and the meta-variables that stands for *smf*: $\overline{f}_1, \overline{f}_2, \overline{g}_1, \overline{g}_2$. The idea is that meta-variables of event terms will be useful to get an abstraction of events in meta-rules, while meta-variables of smf get abstraction of smf appearing in meta-rules.

Definition 3.31 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, $\overline{X}$ the set of meta-variables of terms, the set of meta-terms, $\overline{T}_\Sigma(X, V)$, is the least $S \cup EV$-set defined by:*

$$\begin{array}{ll} 1. & ((T_\Sigma(X,V))_v \cup (\overline{X}_T)_v) \subseteq (\overline{T}_\Sigma(X,V))_v \\ 2. & \left.\begin{array}{ll} t_i \in (\overline{T}_\Sigma(X,V))_{v_i}, i \in \{1,\ldots,n\} & \mathit{and} \\ op \in F, op: v_1 \ldots v_n \rightarrow v & \mathit{then} \end{array}\right\} \Rightarrow op(t_1,\ldots,t_n) \in (\overline{T}_\Sigma(X,V))_v \end{array}$$

The set of $\overline{T}_\Sigma(X, V)$ is the set of all terms that are written with (or without) meta-variables of terms.

Definition 3.32 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, $\overline{X}_{SMF}$, the set of meta-variables of smf, the set of meta-smf, $\overline{SMF}_\Sigma(X, V)$, is the least set defined by:*

$$\begin{array}{ll} 1. & (\overline{X}_{SMF} \cup SMF_\Sigma(X,V)) \subseteq \overline{SMF}_\Sigma(X,V) \\ 2. & \left.\begin{array}{l} \overline{f} \in (\overline{X}_{SMF} \cup SMF_\Sigma(X,V)) \\ \overline{g} \in \overline{SMF}_\Sigma(X,V) \end{array}\right\} \Rightarrow \overline{f}\&\overline{g} \in \overline{SMF}_\Sigma(X,V) \end{array}$$

The set $\overline{SMF}_\Sigma(X, V)$ is the set of all smf written with meta-variables of smf. Note that meta-terms cannot appear in meta-smf.

Definition 3.33 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, $\overline{X}_T$ a set of meta-variables of events, $\overline{X}_{SMF}$ a set of meta-variables of smf, V a set of local states, the set of* Meta-dynamic Axioms *is given by:*

$$\overline{DA}_\Sigma(X,V) = \{\overline{evnt} : \overline{f} \rightarrow \overline{g} \mid \overline{evnt} \in (\overline{T}_\Sigma(X,V))_{ev \in EV}, \overline{f}, \overline{g} \in \overline{SMF}_\Sigma(X,V)\}$$

A meta-dynamic axiom is a dynamic axiom, where meta-variables of terms or meta-variables of smf can appear. Note that meta-variables of terms can only appear in the $\overline{evnt}$ part of the meta-dynamic axiom, and that meta-smf can only appear in the $\overline{f}$ or $\overline{g}$ part of the dynamic axiom.

Definition 3.34 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, $\overline{X}_T$ a set of meta-variables of terms, $\overline{X}_{SMF}$ a set of meta-variables of smf, V a set of local states, the set of* Meta-rules *is given by:*

$$MR_\Sigma(X, V) = \{C_1; \ldots; C_n \quad \leadsto \quad A \,|\, C_i, i = \{1, \ldots, n\}, A \in \overline{DA}_\Sigma(X, V)\}$$

A meta-rule consists of two parts, the left part is made of n Premices (n meta-dynamic axioms), and the right part is one Action (one meta-dynamic axiom). The meta-variables of the Premices stands for syntactical elements, that will be used to derive the new dynamic axiom given by the Action.

Definition 3.35 *From now on, the set X of variables will contain also the meta-variables. For this reason X is then redefined as:*

$$X = X_S \cup X_{EV} \cup \overline{X}_T \cup \overline{X}_{SMF}$$

Example 3.36 *Examples of meta-rules are:*

$$\begin{array}{lcl}
\overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2 \\
\overline{ev}_3 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_4 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & //(\overline{ev}_3, \overline{ev}_4) : \overline{f}_1 \& \overline{g}_1 \rightarrow \overline{f}_2 \& \overline{g}_2 \\
\overline{ev}_3 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_4 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & //(\overline{ev}_3, \overline{ev}_4) : \overline{f}_2 \& \overline{g}_2 \rightarrow \overline{f}_1 \& \overline{g}_1 \\
\overline{ev}_5 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_6 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & //(\overline{ev}_5, \overline{ev}_6) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2
\end{array}$$

See example 3.30 for the types of meta-variables.

These 4 meta-rules explain how work the operator about events: //. The first meta-rule gives the description of // when it works on *ev_nat* sort, // takes two events of natural, symbolized with $\overline{ev}_1, \overline{ev}_2$, whose behavior is given by the Premices $\overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1$ and $\overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2$ and returns a new event of natural whose precondition is given by the concatenation of the precondition of $\overline{ev}_1$ with the precondition of $\overline{ev}_2$, and whose postcondition is given by the concatenation of the postcondition of $\overline{ev}_1$ with the postcondition of $\overline{ev}_2$. This operator can be seen as the "true" parallel operator, in the sense that the resulting event, representing $\overline{ev}_1$, $\overline{ev}_2$ in parallel, behaves like the beginning of $\overline{ev}_1$, followed by the beginning of $\overline{ev}_2$, and it continues with the end of $\overline{ev}_1$ followed by the end of $\overline{ev}_2$. As the arrow $\rightarrow$ represents a local temporal ordering, it is respected in the compound event.

The second and third meta-rules explain how the // operator works over *ev_bool* sorts. There are two meta-rules, that is to say that there are two ways of deriving Dynamic Axioms when this operator is used over *ev_bool* sorts. Firstly, the // operator takes two events of booleans, symbolized with $\overline{ev}_3, \overline{ev}_4$ and returns a new event of booleans whose precondition is the precondition of $\overline{ev}_3$, concatenated with the postcondition of $\overline{ev}_3$ and whose postcondition is the precondition of $\overline{ev}_4$, concatenated with the postcondition of $\overline{ev}_4$. This means that the resulting Dynamic Axiom, explains the behavior of a new event, that works how the event $\overline{ev}_3$ followed by the event $\overline{ev}_4$. The third Meta-Rule is like the second one, except that the roles of $\overline{ev}_3$ and $\overline{ev}_4$ are inversed. These two meta-rules means that the operator // over *ev_bool* sorts behaves like the interleaving: in certain cases the event $\overline{ev}_3$ occurs before, and is followed by $\overline{ev}_4$, and in other cases the event $\overline{ev}_4$ occurs before, and is followed by $\overline{ev}_3$.

The last meta-rule explains the // operator over *ev* sorts. As for the // operator over *ev_nat* sorts, this meta-rule describes the "true" concurrency between two events. As the *ev* event sort is greater than the two other event sorts *ev_nat*, *ev_bool*, this meta-rule produces new events from two other events of sort *ev_nat* or two events of sort *ev_bool* or two events of sort *ev* or two events of different event sorts.

Remark 3.37 *In the example 3.36, the resulting parallel events are obtained in a certain manner ("true" concurrency) for // working over ev_nat sorts and // over ev sort, and in another manner (interleaving) for // working over ev_bool sorts. Since ev is the greatest event sort, compound events of ev_nat sort, ev_bool sort, ev sort or mixed sorts are obtained with the // operator. Another interesting effect is that, if these 4 meta-rules are present, it means that events of sort ev_bool are obtained in two ways: with interleaving or with "true" concurrency.*

Remark 3.38 *Meta-Rules are currently used to easily write Dynamic Axioms, but it is easy to extend this idea to Static Axioms. From some given Static Axioms, we will derive new Static Axioms.*

3.5.4 Set of Axioms

The sets of static, dynamic axioms and meta-rules are collected together to give the set of all axioms of ACCESS.

Definition 3.39 *Let $\Sigma = (S, EV, F)$ be a signature, X a set of variables, V a set of local states, the set of* Axioms *is given by:*

$$AX_\Sigma(X,V) = SA_\Sigma(X,V) \cup DA_\Sigma(X,V) \cup MR_\Sigma(X,V)$$

3.6 Presentation

A system is correctly defined in ACCESS if it is given by a signature, a set of local states, a set of variables and a set of axioms.

Definition 3.40 *A* Presentation *is 4-tuple $PRES = (\Sigma, V, X, Ax)$ where:*

$$\begin{array}{rl}
\Sigma = (S, EV, F) & \textit{is a signature} \\
V & \textit{is a set of local states} \\
X & \textit{is a set of variables} \\
Ax \subseteq AX_\Sigma(X,V) & \textit{is a set of axioms}
\end{array}$$

Remark 3.41 *All Static Axioms appearing in a Dynamic Axiom or in a Meta-rule are called* Static Local Axioms *and must be in the set $SLA_\Sigma(X,V)$. The others, appearing in the Presentation, but neither in Dynamic Axioms nor in Meta-rules are called* Static Global Axioms *and belongs to the set $SA_\Sigma(X,V)$.*

Example 3.42 *An example of presentation for a queue of booleans and a queue of natural numbers is given by the 4-tuple $PRES = (\Sigma, V, X, Ax)$ where:*

$$\begin{array}{rl}
\Sigma = (S, EV, F) & \textit{is the signature of example 3.12} \\
V = \{Nqueue, Bqueue\} & \textit{is the set of local states of example 3.15} \\
X = & \textit{is the set of variables} \\
& \begin{array}{ll}
X_{nat} & = \{m, n, x, y\}, \\
X_{bool} & = \{a, b\}, \\
X_{queue_nat} & = \{X\}, \\
X_{queue_bool} & = \{A\}, \\
X_{ev_nat} & = \emptyset, \\
X_{ev_bool} & = \emptyset, \\
X_{ev} & = \{ev_1, ev_2\}, \\
(\overline{X}_T)_{ev_nat} & = \emptyset, \\
(\overline{X}_T)_{ev_bool} & = \emptyset, \\
(\overline{X}_T)_{ev} & = \{\overline{ev}_1, \overline{ev}_2\}, \\
\overline{X}_{SMF} & = \{\overline{f}_1, \overline{f}_2, \overline{g}_1, \overline{g}_2\}
\end{array}
\end{array}$$

$Ax \subseteq AX_\Sigma(X,V)$ *is the set of axioms, where:*

$$
\begin{array}{rl}
Ax \cap SA_\Sigma(X,V) = \{ & \neg(true) = false \\
& \neg(false) = true \\
& true \vee b = b \\
& false \wedge b = false \\
& a \vee b = \neg(\neg a \wedge \neg b) \\
& pred(succ(n)) = n \\
& n + zero = n \\
& n + succ(m) = succ(n+m) \\
& \neg(D(pred(zero))) \\
& remove\_nat(add\_nat(x, emptyqueue\_nat)) = emptyqueue\_nat \\
& remove\_nat(add\_nat(x, add\_nat(y, X))) = \\
& \qquad add\_nat(x, remove\_nat(add\_nat(y, X))) \\
& first\_nat(add\_nat(x, emptyqueue\_nat)) = x \\
& first\_nat(add\_nat(x, add\_nat(y, X))) = \\
& \qquad first\_nat(add\_nat(y, X)) \\
& \neg(D(remove\_nat(emptyqueue\_nat))) \\
& remove\_bool(add\_bool(a, emptyqueue\_bool)) = emptyqueue\_bool \\
& remove\_bool(add\_bool(a, add\_bool(b, A))) = \\
& \qquad add\_bool(a, remove\_bool(add\_bool(b, A))) \\
& first\_bool(add\_bool(a, emptyqueue\_bool)) = a \\
& first\_bool(add\_bool(a, add\_bool(b, A))) = \\
& \qquad first\_bool(add\_bool(b, A)) \\
& \neg(D(remove\_bool(emptyqueue\_bool))) \\
& //(ev_1, ev_2) = //(ev_2, ev_1) \\
& //(ev_1, //(ev_2, ev_3)) = //(//(ev_1, ev_2), ev_3)\} \\
Ax \cap DA_\Sigma(X,V) = \{ & get\_nat(m): \quad \neg(Nqueue = emptyqueue\_nat) \rightarrow \\
& \qquad\qquad (m = first\_nat(Nqueue))\& \\
& \qquad\qquad Nqueue := remove\_nat(Nqueue) \\
& put\_nat(m): \quad \epsilon \rightarrow Nqueue := add\_nat(m, Nqueue) \\
& get\_bool(a): \quad \neg(Nqueue = emptyqueue\_bool) \rightarrow \\
& \qquad\qquad (a = first\_bool(Nqueue))\& \\
& \qquad\qquad Nqueue := remove\_bool(Nqueue) \\
& put\_bool(a): \quad \epsilon \rightarrow Nqueue := add\_bool(a, Nqueue)\} \\
Ax \cap MR_\Sigma(X,V) = & \{\overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \quad \rightsquigarrow \quad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2\}
\end{array}
$$

This presentation describes the behavior of two queues of natural and booleans respectively. It consists of the signature of example 3.12, that gives the sorts and the operations over the sorts. The presentation contains also two local states $\{Nqueue, Bqueue\}$, one for each queue, whose sorts are $queue\_nat$ and $queue\_bool$ respectively. The set of variables collects all variables used in the axioms. The static axioms of the presentation are constraints about the operations of the signature, we have Equational Atom about natural, booleans, queues of natural, queues of booleans and events. There are three Predicates, one about $pred(zero)$ and the others about $emptyqueue\_nat$, $emptyqueue\_bool$. The equational atoms about events express the commutativity and the associativity of the $//$ operator over ev sorts. The dynamic axioms explain the behavior of some events and the meta-rule describes the true concurrency for events produced with the $//$ operator. See example 3.28 for the dynamic axioms.

Chapter 4

Semantics of Access

The semantics of an ACCESS presentation takes its values in a structure: an algebra for the signature of the presentation together with a transition system defined over this algebra. The algebra contains the possible values of the data sorts S of the presentation, the values of the events EV and the functions corresponding to the operators of F working on the data and events. The transition system is a set of triples, each of them giving an initial state, a transition changing the state, and a final state.

The structure is a model of the system if the constraints of the presentation are satisfied: the global static axioms have to be satisfied for all possible states of the system; for each triple of the transition system, there must be a dynamic axiom in the presentation that explains how the final state of the triple is reached, when the transition occurs during the initial state.

This chapter presents the way to derive all possible dynamic axioms from a given set of Meta-rules. We will then give the definition of structures corresponding to ACCESS Presentations, followed by the evaluation of all syntactical elements of the Presentation in the structure. Finally, we will explain which structures are models.

4.1 Dynamic Axioms derived from Meta-rules

Meta-Rules have been introduced in order to simplify the writing of Dynamic Axioms. The idea is that for each set of Dynamic Axioms that fits the n Premices (n meta-dynamic axioms) of a Meta-rule, we can derive a new Dynamic Axiom by the Action part of the Meta-rule.

A Dynamic Axiom fits a meta-dynamic axiom if all meta-variables appearing in the meta-dynamic axiom can be substituted by syntactical elements (terms or smf) such that the meta-dynamic axiom, with the syntactical elements in place of the meta-variables, is equal to the given Dynamic Axiom.

The derived Dynamic Action is then given by substituting in the Action part of the Meta-rule the meta-variables with the syntactical elements used in the premices part of the Meta-rule.

Definition 4.1 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, the substitution of the meta-variables of terms is one of the functions $\overline{I}$ of the set:*

$$\overline{subst}_{X_T} = \{\overline{I} : \overline{X}_T \rightarrow T_\Sigma(X, V) \mid \overline{I} \textit{ is a } (S \cup EV) - \textit{morphism}\}$$

This substitution associates to each meta-variable of terms one term $t \in T_\Sigma(X, V)$. Remember that t is a syntactical element, it is not a value in an algebra.

Definition 4.2 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, the substitution of the meta-variables of smf is one of the functions $\overline{J}$ of the set:*

$$\overline{subst}_{X_{SMF}} = \{\overline{J} : \overline{X}_{SMF} \rightarrow SMF_\Sigma(X, V) \mid \overline{J} \textit{ is a total function}\}$$

Similarly to the meta-variables of terms, the meta-variables of smf are associated to a $smf \in SMF_\Sigma(X, V)$ by the mean of a substitution.

Definition 4.3 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, $\overline{I}$ a substitution for meta-variable of terms, the evaluation of the meta-terms is given by the function:*

$$\overline{eval}_{T,\overline{I}} : \overline{T}_\Sigma(X, V) \to T_\Sigma(X, V)$$

defined by:

$$\begin{array}{l}
\overline{eval}_{T,\overline{I}}(t) = t, t \in T_\Sigma(X, V) \\
\overline{eval}_{T,\overline{I}}(\overline{x}) = \overline{I}(\overline{x}), \overline{x} \in \overline{X}_T \\
\overline{eval}_{T,\overline{I}}(op(t_1), \ldots, op(t_n)) = op(\overline{eval}_{T,\overline{I}}(t_1), \ldots, \overline{eval}_{T,\overline{I}}(t_n)), t_i \in \overline{T}_\Sigma(X, V)
\end{array}$$

A meta-term is evaluated to a term where all syntactical elements of the meta-terms remain unchanged and where all the meta-variables are replaced by the syntactical term associated to it by the substitution.

Definition 4.4 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, $\overline{J}$ a substitution for meta-variables of smf, the evaluation of the meta-smf is given by the function:*

$$\overline{eval}_{SMF,\overline{J}} : \overline{SMF}_\Sigma(X, V) \to SMF_\Sigma(X, V)$$

defined by:

$$\begin{array}{l}
\overline{eval}_{SMF,\overline{J}}(smf) = smf, smf \in SMF_\Sigma(X, V) \\
\overline{eval}_{SMF,\overline{J}}(\overline{smf}) = \overline{J}(\overline{smf}), \overline{smf} \in \overline{X}_{SMF} \\
\overline{eval}_{SMF,\overline{J}}(f \& g) = \overline{eval}_{SMF,\overline{J}}(f) \ \& \ \overline{eval}_{SMF,\overline{J}}(g), f, g \in \overline{SMF}_\Sigma(X, V)
\end{array}$$

A meta-smf is evaluated to a smf where all syntactical elements of the meta-smf remain unchanged and where all meta-smf are changed to a smf given by the substitution.

Definition 4.5 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, $\overline{I}$ a substitution for meta-variables of terms, $\overline{J}$ a substitution for meta-variables of smf, the evaluation of the meta-dynamic axioms is given by the function:*

$$\overline{eval}_{DA,\overline{I},\overline{J}} : \overline{DA}_\Sigma(X, V) \to DA_\Sigma(X, V)$$

defined by:

$$\overline{eval}_{DA,\overline{I},\overline{J}}(\overline{evnt} : \overline{f} \to \overline{g}) = \overline{eval}_{T,\overline{I}}(\overline{evnt}) : \overline{eval}_{SMF,\overline{J}}(\overline{f}) \to \overline{eval}_{SMF,\overline{J}}(\overline{g})$$

A dynamic axiom is derived from a meta-dynamic axiom by writing the meta-dynamic axiom with the syntactical elements, given by the two substitutions of terms and smf, in place of the meta-variables.

Free variables (not meta-) with the same name, appearing in two or more premices, of a meta-rule are supposed to be different variables. For that reason, we introduce the renaming of free variables, in order to have separate sets of variable names between the premices.

Definition 4.6 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, a renaming of variables, σ, is a total $(S \cup EV)$-morphism:*

$$\sigma : X_S \cup X_{EV} \to X_S \cup X_{EV}$$

Remember that a $(S \cup EV)$-morphism is a total function m between two $(S \cup EV)$-sets, E_1, E_2 such that $m((E_1)_v) \subseteq (E_2)_v$, $v \in (S \cup EV)$.

Definition 4.7 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, σ a renaming of variables, the extension of the renaming σ to the terms $t \in T_\Sigma(X, V)$ is a $(S \cup EV)$-morphism:*

$$\sigma_T : T_\Sigma(X, V) \to T_\Sigma(X, V)$$

defined by:

$$\sigma_T(x) = \sigma(x), x \in X_S \cup X_{EV}$$
$$\sigma_T(v) = v, v \in V$$
$$\sigma_T(op(t_1,\ldots,t_n)) = op(\sigma_T(t_1),\ldots,\sigma_T(t_n)), \left(\begin{array}{l} op \in F, op: v_1 \ldots v_n \to v \text{ and} \\ t_i \in (T_\Sigma(X,V))_{v_i}, i \in \{1,\ldots,n\} \end{array} \right)$$

The extension of a renaming of variables to the terms is nothing else than the same term with renamed variables.

Remark 4.8 *As we defined it for terms, we define the extension of a renaming of variables to Static and Dynamic Axioms as the same Static or Dynamic axioms with corresponding renamed variables in place of their variables.*

Remark 4.9 *From now on, extensions of renamings to terms, σ_T, and to Static and Dynamic Axioms will all be called, σ, since no confusion is possible.*

Definition 4.10 *Let $PRES = (\Sigma, V, X, Ax)$ be a presentation, the set of derived Dynamic Axioms, $dDA_\Sigma(X,V)$, is the least set defined by:*

1. $Ax \cap DA_\Sigma(X,V) \subseteq dDA_\Sigma(X,V)$
2. $$\left. \begin{array}{l} \left(\begin{array}{l} mr = C_1;\ldots;C_n \quad \leadsto \quad A \in MR_\Sigma(X,V) \\ da_i \in dDA_\Sigma(X,V), i = \{1,\ldots,n\} \end{array} \right), \textit{such that:} \\ \quad \exists \sigma \textit{ a renaming of variables} \\ \quad \exists \overline{I} \in \overline{subst}_{X_T}, \overline{J} \in \overline{subst}_{X_{SMF}} \\ \quad \textit{with:} \\ \quad Var(\sigma(da_i)) \cap Var(\sigma(da_j)) = \emptyset, \forall i,j \in \{1,\ldots,n\}, i \neq j \\ \quad \overline{eval}_{DA,\overline{I},\overline{J}}(C_i) = \sigma(da_i), \end{array} \right\} \Rightarrow \overline{eval}_{DA,\overline{I},\overline{J}}(A) \in dDA_\Sigma(X,V)$$

A derived Dynamic Axiom is either a (usual) Dynamic Axiom, i.e. an element of $Ax \cap DA_\Sigma(X,V)$, or an actually derived Dynamic Axiom. Such a Dynamic Axiom is obtained from a Meta-rule by fixing the meta-variables with the Premices, C_i, and by replacing them in the Action of the meta-rule. Meta-variables are fixed, when two substitutions $\overline{I}, \overline{J}$ of terms and smf have been found, such that for each Premice C_i, there exists a corresponding Dynamic Axiom da_i that can be considered as derived from C_i, with these two substitutions. The renaming of variables, σ, is used when two or more da_i have common free variables of terms. In this case, their free variables are renamed such that there is no more common free variables, i.e. $Var(\sigma(da_i)) \cap Var(\sigma(da_j)) = \emptyset, \forall i,j \in \{1,\ldots,n\}, i \neq j$. The resulting Dynamic Axiom is then produced with the $\sigma(da_i)$, instead of the da_i. It exists always a renaming producing different free variables among the da_i, even if new variables have to be added to the set X of variables.

If two dynamic axioms share the same free variable, n, it means that the value associated to this variable is meant to be the same. Usually, free variables in two different dynamic axioms represent two different values (even if the variables have casually the same name). Systematic renamings avoid common free variables between dynamic axioms involved in a meta-rule.

Note that the same Dynamic Axiom can appear in two or more da_i.

From now on, meta-rules and meta-variables will never be used. The word *variable* will be employed for variables of $X_S \cup X_{EV}$ excluding variables of $\overline{X}_T \cup \overline{X}_{SMF}$.

Definition 4.11 *Given $PRES = (\Sigma, V, X, Ax)$ a presentation, the derived presentation is given by $PRES' = (\Sigma, V, X, Ax')$, where $Ax' = Ax \setminus (Ax \cap MR_\Sigma(X,V)) \cup dDA_\Sigma(X,V)$.*

Remark 4.12 *The premices (meta-dynamic axioms) of a meta-rule may have free variables. If two occurrences of the same variable (not meta-variable) appear into two different premices of the same meta-rule, these two occurrences become two different variables by the mean of a renaming. However, in* *some cases, the same names of variables are needed in different premices, in order to represent the same value. This is realized by the use of meta-variables of terms instead of variables. As meta-variables are not submitted to renamings, the use of the same meta-variable inside two different premices means that it represents the same syntactical element.*

Example 4.13 *Given the Meta-rule of example 3.36:*

$$\overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \quad \rightsquigarrow \quad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2$$

and the two Dynamic Axioms:

$$\begin{aligned} get\_nat(m) : \;& \neg(Nqueue = emptyqueue\_nat) \rightarrow \\ & (m = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \\ put\_nat(m) : \;& \epsilon \rightarrow Nqueue := add\_nat(m, Nqueue) \end{aligned}$$

of example 3.28, we can derive new Dynamic Axioms. Using renamings we have:

$$\begin{aligned} //(get\_nat(m), get\_nat(n)) : \;& \neg(Nqueue = emptyqueue\_nat) \& \neg(Nqueue = emptyqueue\_nat) \rightarrow \\ & (m = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \& \\ & (n = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \end{aligned}$$

$$\begin{aligned} //(get\_nat(m), put\_nat(n)) : \;& \neg(Nqueue = emptyqueue\_nat) \& \epsilon \rightarrow \\ & (m = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \& \\ & Nqueue := add\_nat(n, Nqueue) \end{aligned}$$

$$\begin{aligned} & //(//(get\_nat(m), get\_nat(n)), put\_nat(n')) : \\ & \qquad \neg(Nqueue = emptyqueue\_nat) \& \\ & \qquad \neg(Nqueue = emptyqueue\_nat) \& \epsilon \rightarrow \\ & \qquad (m = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \& \\ & \qquad (n = first\_nat(Nqueue)) \& Nqueue := remove\_nat(Nqueue) \& \\ & \qquad Nqueue := add\_nat(n', Nqueue) \end{aligned}$$

$$//(//(//(\ldots))) : \ldots$$

An infinity of derived dynamic axioms are created in that way with this meta-rule.

4.2 Structure

A structure is given by both an algebra of the signature and a transition system. The algebra is a set of values for both the data and the events, together with functions over these values corresponding to the operators of the signature working over sorts and events. The transition system is a set of triples: ((initial) global state, transition, (final) global state). The transition occurs during the initial global state of the system and produces the final global state.

Definition 4.14 *Let* $PRES = (\Sigma, V, X, Ax)$ *be a presentation, a* Σ*-structure is a pair* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *where:*

1. A_Σ *is a pair* $A_\Sigma = (A, F_A)$ *and:*
   A *is a* $(S \cup EV)$*-set such that:*
   $$\begin{aligned} \forall(s_1, s_2) &\in \subseteq_S &\Rightarrow A_{s_1} &\subseteq A_{s_2} \\ \forall(ev_1, ev_2) &\in \subseteq_{EV} &\Rightarrow A_{ev_1} &\subseteq A_{ev_2} \end{aligned}$$
   $F_A = \{op_A : A_{v_1} \times \ldots \times A_{v_n} \rightarrow A_v \mid op_A$ *is a partial function and* $\exists op : v_1 \ldots v_n \rightarrow v, op \in F\}$
   such that:
   $$\left.\begin{array}{l} op_A : A_{v'_1} \times \ldots \times A_{v'_n} \rightarrow A_{v'} \in F \\ op_A : A_{v_1} \times \ldots \times A_{v_n} \rightarrow A_v \in F \\ (v'_1 \ldots v'_n, v_1, \ldots v_n) \in \subseteq_{S \cup EV} \end{array}\right\} \Rightarrow op_{A_{v_1 \ldots v_n, v}}|_{A_{v'_1} \times \ldots \times A_{v'_n}} = op_{A_{v'_1 \ldots v'_n, v'}}$$
2. $TRS_\Sigma \subseteq (V_A \times A_{(ev \in EV)} \times V_A)$

A_Σ is the algebra part of the signature. It is made of a $(S \cup EV)$-set of values, A, and of a set, F_A, of partial functions op_A, one for each operators $op \in F$. Remember that F is the set of operations given by the signature $\Sigma = (S, EV, F)$ and $\subseteq_S$, $\subseteq_{EV}$ are the partial order of the signature, and $\subseteq_{S \cup EV}$ is the extension of these two order relations to the strings of equal length in $(S \cup EV)^*$. These orderings induce certain conditions on the Σ-structure: (1) the carrier sets A_v have to respect the two partial orders, $\subseteq_S$, $\subseteq_{EV}$: the carrier set of a supersort has to contain the carrier sets of its subsorts; (2) the restriction of an overloading function to subsorts (i.e. sorts of the overloaded function) must be equal to the overloaded function itself. See notation 3.7 of the operations.

TRS_Σ is the transition system, it consists of triples made of two global states and one transition. A transition is one of the possible values for event sorts, i.e. a transition belongs to $A_{(ev \in EV)}$. A global state is a collection of values: one for each local state, see definition 4.20 below.

Example 4.15 *Let $PRES = (\Sigma, V, X, Ax)$ be the presentation of example 3.42, a structure for this presentation is given by $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ where:*

1. $A_\Sigma = (A, F_A)$ *and:*

$$A = A_{nat} \cup A_{bool} \cup A_{queue\_nat} \cup A_{queue\_bool} \cup A_{ev\_nat} \cup A_{ev\_bool} \cup A_{ev}$$

$$\begin{array}{ll}
A_{nat} & = \mathcal{N} \\
A_{bool} & = \{T, F\} \\
A_{queue_nat} & = \{\ [], [0], [1], [2], \ldots, [0,0], [0,1], [0,2], \ldots, [1,0], [1,1], [1,2], \ldots, \\
& \quad [0,0,1], [0,0,2], \ldots\} \\
A_{queue_bool} & = \{\ [], [T], [F], [T,F], [F,T], [T,T,F], [T,F,T], \\
& \quad [F,T,T], [F,T,F], [F,F,T], \ldots\} \\
A_{ev_nat} & = \{\ put_0, put_1, put_2, \ldots, put_{0,0}, put_{0,1}, put_{0,2}, \ldots, get_0, get_1, get_2, \ldots, \\
& \quad get_{0,0}, get_{0,1}, get_{0,2}, \ldots, misc1_{nat}, misc2_{nat}, \ldots\} \\
A_{ev_bool} & = \{\ put_T, put_F, put_{T,F}, put_{F,T}, put_{T,T,F}, put_{T,F,T}, \ldots get_T, get_F, get_{T,F}, \\
& \quad get_{F,T}, get_{T,T,F}, get_{T,F,T}, \ldots, misc1_{bool}, misc2_{bool}, \ldots\} \\
A_{ev} & = A_{ev_nat} \cup A_{ev_bool} \cup \{misc1, misc2, misc3, \ldots\}
\end{array}$$

$$
\begin{array}{llll}
F_A = \{ & ZERO: & & \rightarrow \mathcal{N}, \\
& PRED: & \mathcal{N} & \rightarrow \mathcal{N}, \\
& SUCC: & \mathcal{N} & \rightarrow \mathcal{N}, \\
& +: & \mathcal{N} \times \mathcal{N} & \rightarrow \mathcal{N}, \\
& TRUE: & & \rightarrow \{T, F\}, \\
& FALSE: & & \rightarrow \{T, F\}, \\
& \neg: & \{T, F\} & \rightarrow \{T, F\}, \\
& \vee: & \{T, F\} & \rightarrow \{T, F\}, \\
& \wedge: & \{T, F\} & \rightarrow \{T, F\}, \\
& EMPTYQUEUE\_N: & & \rightarrow A_{queue\_nat}, \\
& ADD\_N: & \mathcal{N} \times A_{queue\_nat} & \rightarrow A_{queue\_nat}, \\
& REMOVE\_N: & A_{queue\_nat} & \rightarrow A_{queue\_nat}, \\
& FIRST\_N: & A_{queue\_nat} & \rightarrow \mathcal{N}, \\
& EMPTYQUEUE\_B: & & \rightarrow A_{queue\_bool}, \\
& ADD\_B: & \{T, F\} \times A_{queue\_bool} & \rightarrow A_{queue\_bool}, \\
& REMOVE\_B: & A_{queue\_bool} & \rightarrow A_{queue\_bool}, \\
& FIRST\_B: & A_{queue\_bool} & \rightarrow \{T, F\}, \\
& GET\_N: & \mathcal{N} & \rightarrow A_{ev\_nat}, \\
& PUT\_N: & \mathcal{N} & \rightarrow A_{ev\_nat}, \\
& GET\_B: & \{T, F\} & \rightarrow A_{ev\_bool}, \\
& PUT\_B: & \{T, F\} & \rightarrow A_{ev\_bool}, \\
& //: & A_{ev\_nat} \times A_{ev\_nat} & \rightarrow A_{ev\_nat}, \\
& //: & A_{ev\_bool} \times A_{ev\_bool} & \rightarrow A_{ev\_bool}, \\
& //: & A_{ev} \times A_{ev} & \rightarrow A_{ev}\}
\end{array}
$$

2. $$
\begin{array}{ll}
TRS_\Sigma = \{ & (([1,2],[T,F,T]), put_{0,1}, ([1,0,1,2],[T,F,T])), \\
& (([1,2],[T,F,T]), put_F, ([1,2],[F,T,F,T])), \\
& (([1,2],[T,F,T]), misc1, ([1,0,1,2],[F,T,F,T])), \\
& (([1,1,2],[F,T,F,T]), misc1, ([1,1,0,1,2],[F,F,T,F,T])), \\
& (([2,3],[]), put_{1,0}, ([0,1,2,3],[])), \\
& (([2,3],[]), put_{0,1}, ([1,0,2,3],[])), \\
& (([0,1],[]), put_{0,1}, ([1,0,0,1],[])), \\
& (([0,1],[]), get_{0,1}, ([],[])), \\
& \ldots\}
\end{array}
$$

The algebra part A_Σ of the structure of this example is given by the pair $A_\Sigma = (A, F_A)$ where A is made of 7 sets, one for each sort of the presentation. The carrier sets of *nat*, *bool* sorts are the usual sets of natural numbers, and booleans: $\mathcal{N}$ and $\{T, F\}$ respectively. The set of queues of natural numbers is made of ordered list of natural, e.g. $[1, 2]$ is the queue where the first element is 2 and the second and last one is 1. The set of queues of booleans is similar, except that natural numbers are replaced with elements in $\{T, F\}$. The next two sets are made of events occurring on queues of natural and queues of booleans respectively. As an example, $get_{0,1}$ is an event on queues of natural, this event consists in extracting the element 0 from a queue and then an element 1 from the queue; $put_{0,1}$ is an event on queues of natural, this event consists in putting the element 0 into a queue and then the element 1 into the queue. The *misc* events are complex events producing new queues from a given queue, and that in different manners, e.g. by mixing *get* and *put* operations . The last set is made of events of *ev* sort, e.g. mixing *put*, *get* for naturals with *put*, *get* for booleans. The partial orders of the signature of the given presentation are respected, as we have $A_{ev\_nat} \subseteq A_{ev}$ and $A_{ev\_bool} \subseteq A_{ev}$.

The set F_A of A_Σ contains usual operations over natural numbers and booleans, as well as usual operations on queues. Operations producing events are for example GET_N, defined by: $GET\_N(i) = get_i$. The $//$ operation produces new events for example: $//(get_1, put_0) = misc1_{nat}$. For each $op \in F$ there is a corresponding $op_A \in F_A$. Some operations of F_A are partial, it is the case for $PRED$,

$REMOVE\_N$, $REMOVE\_B$, $FIRST\_N$, $FIRST\_B$, which are not defined over 0 for $PRED$ and over emptyqueues for the others.

The transition system of the structure is made of several triples, for example the triple: $(([1,2],[T,F,T]), put_{0,1}, ([1,0,1,2],[T,F,T]))$ stands for the initial global state $([1,2],[T,F,T])$ that becomes the final global state $([1,0,1,2],[T,F,T])$ when the event $put_{0,1}$ has occurred. A global state is a pair of two values one for each local state. Remember that the given presentation contains two local states: $Nqueue$ of sort $queue\_nat$ and $Bqueue$ of sort $queue\_bool$.

The next sections explain how a syntactical element of the Presentation has an associated value in the Σ-structure.

Variables, Local States and Terms are evaluated to values in the algebra. Axioms, either Static or Dynamic will not be evaluated to a value in the algebra, but to a Change of Global State. The reason for this is that Dynamic Axioms are used to explain how global states *change* when an event occurs. The evaluation of a Dynamic Axiom is then a Change of Global State, giving for each current global state, a new global state resulting of the occurrence of the event defined by the Dynamic Axiom. As Static Axioms appear in the writing of Dynamic Axioms, it is convenient to evaluate also Static Axioms to Changes of Global States.

4.3 Interpretation of Variables

Variables are interpreted in the algebra A_Σ by the mean of a function giving for each variable a (unique) value in the algebra. The value and the variable have to be of the same sort. Meta-variables are not concerned.

Definition 4.16 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, the interpretation of the variables $x \in (X_S \cup X_{EV})$ is one of the (total) functions I of the set $Interp_X$ where:*

$$Interp_X = \{I : X_S \cup X_{EV} \rightarrow A \,|\, I \textit{ is } a\,(S \cup EV) - morphism\}$$

Definition 4.17 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, I an interpretation of the variables, $x \in (X_S \cup X_{EV})$ a variable, the interpretation changing only one variable is one of the functions I' of the set:*

$$Interp_{X,I,x} = \{I' \in Interp_X \,|\, : \; I'(y) = I(y), y \neq x\}$$

Given I an interpretation of variables and x a variable, a function I' of $Interp_{X,I,x}$ is an interpretation of variables that associates to each variable the same value than I except for the variable x.

Example 4.18 *Given the presentation $PRES = (\Sigma, V, X, Ax)$ of example 3.42, and the Σ-structure $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ of example 4.15, an interpretation I of the set of variables is given by:*

$$\begin{array}{l} I(m) = 2, I(n) = 3, I(x) = 4, I(y) = 1 \\ I(a) = T, I(b) = F \\ I(X) = [1,0,3,12,5], I(A) = [T,F,F,T] \\ I(ev_1) = misc1, I(ev_2) = put_{T,F} \end{array}$$

4.4 Interpretation of Local States

Similarly to Variables, Local states are evaluated in the algebra by a function associating to each local state a value in the algebra. Naturally, the sorts of the local states and their associated value have to be equal.

Definition 4.19 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, the interpretation of the Local States $v \in V$ is one of the partial functions st of the set V_A where:*

$$V_A = \{st : V \rightarrow A \,|\, st \textit{ is } a\,S - function\}$$

The interpretation of Local States is a mapping that associates a value (of the right sort) to each Local State. Remember that a S-function is a partial S-morphism.

Definition 4.20 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, a Global State is a partial S-function:*

$$st : V \rightarrow A$$

Remark 4.21 *V_A is the set of the Global States.*

A Global State is nothing else but a function associating to each Local State a value in the algebra. The Global State is then the set of values associated to the set of Local States.

Definition 4.22 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, st a Global State, $v \in V$ a local state, the definedness predicate of v, $D_{V,st}(v)$, is given by:*

$$D_{V,st}(v) \Leftrightarrow st(v) exists$$

Example 4.23 *Given the presentation $PRES = (\Sigma, V, X, Ax)$ of example 3.42, and the Σ-structure $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ of example 4.15, some global states can be given by:*

$$\begin{array}{l} st_1(Nqueue) = [1,2,0,2], st_1(Bqueue) = [T,F,F,T] \\ st_2(Nqueue) = [], st_2(Bqueue) = [T] \\ st_3(Nqueue) = [9,2] st_4(Nqueue) = [0,1], st_4(Bqueue) = [] \end{array}$$

These global states give different values to each local state, except st_3 that is undefined for $Bqueue$.

Definition 4.24 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, a Change of Global State is a partial function:*

$$cst : V_A \rightarrow V_A$$

The set of the Changes of Global State is the set:

$$CST_A = \{cst : V_A \rightarrow V_A \,|\, cst \text{ is a partial function}\}$$

A Change of Global State is a partial function that associates to each Global State another Global State.

Example 4.25 *Given the presentation $PRES = (\Sigma, V, X, Ax)$ of example 3.42, and the Σ-structure $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ of example 4.15, and the global states of example 4.23, some Changes of Global State are given by:*

$$\begin{array}{l} cst_1(st_1) = st_3, cst_1(st_2) = st_3, cst_1(st_3) = st_3 \\ cst_2(st_1) = st_2, cst_2(st_2) = st_3, cst_2(st_3) = st_4, cst_2(st_4) = st_1 \end{array}$$

With the Change of Global state cst_1 each global state is changed to the global state st_3 except st_4, for which the change is not defined. With the Change of Global state cst_2, each global state is changed to another global state forming a cycle.

4.5 Evaluation of Terms

The interpretation of Terms depends on the interpretation of variables and on the current global state of the system.

Definition 4.26 *Let* $PRES = (\Sigma, V, X, Ax)$ *be a Presentation,* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *a* Σ*-structure,* $I \in Interp_X$ *an interpretation function for variables, the evaluation of the Terms* $t \in T_\Sigma(X,V)$ *is given by the partial function:*

$$[[\_]]_{T,I} : V_A \times T_\Sigma(X,V) \rightarrow A$$

defined by:

1. $[[st, v]]_{T,I} = \begin{cases} st(v), \text{ if } v \in V \text{ and } st(v) \text{ exists} \\ undefined, \text{ otherwise} \end{cases}$
2. $[[st, x]]_{T,I} = I(x), \forall x \in (X_S \cup X_{EV})$
3. $[[st, op(t_1, \ldots, t_n)]]_{T,I} = \begin{cases} op_A([[st, t_1]]_{T,I}, \ldots, [[st, t_n]]_{T,I}), \left(\begin{array}{l} \text{if } op : v_1 \ldots v_n \rightarrow v \text{ and} \\ \forall i \in \{1, \ldots, n\}, [[st, t_i]]_{T,I} \text{ exists} \end{array} \right) \\ undefined, \text{ otherwise} \end{cases}$

The evaluation of a term depends on the given interpretation of variables, I, and on the current state st.

A Local State is evaluated to its value in the Global State. A variable is evaluated to its interpretation by I. A more complex term is evaluated by functions in F_A applied upon inmost evaluated terms.

Definition 4.27 *Let* $PRES = (\Sigma, V, X, Ax)$ *be a Presentation,* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *a* Σ*-structure, st a Global State,* $t \in T_\Sigma(X,V)$ *a Term, the definedness predicate of t,* $D_{T,I,st}(t)$*, is given by:*

$$D_{T,I,st}(t) \Leftrightarrow [[st, t]]_{T,I} exists$$

Remark 4.28 *As the evaluation functions are* partial *ones, we use definedness predicates to assert if the function is, or isn't, defined for a term (later for a Static or Dynamic Axiom).*

Don't confuse these definedness predicates, useful to explain the semantics, with the Predicates for terms defined in the syntax, which are used to specify properties of the system, as for example, the impossibility to have a given event under special conditions.

Example 4.29 *Given the presentation* $PRES = (\Sigma, V, X, Ax)$ *of example 3.42, and the* Σ*-structure* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *of example 4.15, with the global state* st_1 *of example 4.23 and the interpretation of variables of example 4.18, we have the following evaluation for some terms:*

$$\begin{array}{l}
[[st_1, Nqueue]]_{T,I} = st_1(Nqueue) = [1,2,0,2] \\
[[st_1, X]]_{T,I} = I(X) = [1,0,3,12,5] \\
[[st_1, get_nat(succ(x))]]_{T,I} \\
\qquad = GET_N([[st_1, succ(x)]]_{T,I}) \\
\qquad = GET_N(SUCC([[st_1, x]]_{T,I})) \\
\qquad = GET_N(SUCC(I(x))) = GET_N(SUCC(4)) = GET_N(5) = get_5 \\
[[st_1, //(get_nat(m), get_nat(n))]]_{T,I} \\
\qquad = //([[st_1, get_nat(m)]]_{T,I}, [[st_1, get_nat(m)]]_{T,I}) \\
\qquad = //(GET_N(2), GET_N(3)) = get_{2,3}
\end{array}$$

4.6 Evaluation of Static Axioms

A Static Axiom (local or global) is well defined if it is satisfied in the algebra. To do this all the terms appearing in the axiom are evaluated in the algebra, the terms are then replaced by their value in the axiom. The axiom is then evaluated with these values. If the evaluation is true then the definedness predicate associated to the axiom is said to be satisfied. Finally, the Static Axiom is evaluated to a Change of Global State, this change is nothing else but the identity (i.e. the global state remains unchanged) if the Static Axiom is well defined. In the contrary, the evaluation of the Static Axiom is undefined.

Definition 4.30 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, $I \in Interp_X$ an interpretation function for variables, st a global state, the definedness predicate of a Static Axiom is given by:*

1. $\forall t_1, t_2 \in (T_\Sigma(X,V))_{(s \in S)}$:
$$D_{SA,I,st}(t_1 = t_2) \Leftrightarrow (D_{T,I,st}(t_1) \wedge D_{T,I,st}(t_2) \wedge [[st, t_1]]_{T,I} = [[st, t_2]]_{T,I}) \vee (\neg D_{T,I,st}(t_1) \wedge \neg D_{T,I,st}(t_2))$$
2. $\forall t_1, t_2 \in (T_\Sigma(X,V))_{(ev \in EV)}$:
$$D_{SA,I,st}(t_1 = t_2) \Leftrightarrow \begin{array}{l} (D_{T,I,st}(t_1) \wedge D_{T,I,st}(t_2) \wedge [[st, t_1]]_{T,I} = [[st, t_2]]_{T,I}) \vee \\ (\neg D_{T,I,st}(t_1) \wedge \neg D_{T,I,st}(t_2)) \vee \\ \left(\begin{array}{l} (\forall (st, w, b) \in TRS_\Sigma \text{ s.t. } [[st, t_1]]_{T,I} = w \Rightarrow \\ \qquad \exists (st, w', b) \in TRS_\Sigma \text{ s.t. } [[st, t_2]]_{T,I} = w') \wedge \\ (\forall (st, w', b) \in TRS_\Sigma \text{ s.t.} [[st, t_2]]_{T,I} = w' \Rightarrow \\ \qquad \exists (st, w, b) \in TRS_\Sigma \text{ s.t. } [[st, t_1]]_{T,I} = w) \end{array} \right) \end{array}$$
3. $\forall t \in T_\Sigma(X,V)$:
$$D_{SA,I,st}(D(t)) \Leftrightarrow D_{T,I,st}(t)$$
4. $\forall sa, sa_1, sa_2 \in SA_\Sigma(X,V)$:
$$\begin{array}{lcl} D_{SA,I,st}(sa_1 \wedge sa_2) & \Leftrightarrow & D_{SA,I,st}(sa_1) \wedge D_{SA,I,st}(sa_2) \\ D_{SA,I,st}(sa_1 \vee sa_2) & \Leftrightarrow & D_{SA,I,st}(sa_1) \vee D_{SA,I,st}(sa_2) \\ D_{SA,I,st}(sa_1 \Rightarrow sa_2) & \Leftrightarrow & D_{SA,I,st}(sa_1) \Rightarrow D_{SA,I,st}(sa_2) \\ D_{SA,I,st}(\neg sa) & \Leftrightarrow & \neg D_{SA,I,st}(sa) \\ D_{SA,I,st}(\exists . x(sa)) & \Leftrightarrow & \exists I' \in Interp_{X,I,x} D_{SA,I',st}(sa) \\ D_{SA,I,st}(\forall . x(sa)) & \Leftrightarrow & \forall I' \in Interp_{X,I,x} D_{SA,I',st}(sa) \end{array}$$

An Equational Atom between terms of data sorts is well defined if the two data terms of the Atom have the same evaluation in the algebra or if the two data terms are undefined. This is the strong equality of [Broy82].

An Equational Atom between terms of event sorts is well defined if the two event terms have the same evaluation in the algebra or if they are both undefined or if the two event terms evaluate in two (different) transitions w, w' of the algebra (elements of $(A)_{ev \in EV}$) whose behavior is identical. Two such transitions have the same behavior if for each pair of triple, in the transition system TRS_Σ, whose initial global state is the current state st, whose transition is w (or w'), whose final global state is b, then there exists another triple, whose initial global state is the current state st, whose transition is w' (or w), and whose final global state is the same global state b. This condition means that for each triple where w is present there must be another identical triple where w' takes the place of w and vice-versa. The behavior of two events is then identical if they evaluate into two transitions, whose occurrences on the current global state produce the same final global state.

A Predicate for a term is well defined if the evaluation of the term exists.

The combination of Static Axioms with the logical connectors are well defined if the combination of the corresponding definedness predicates are well defined.

Remark 4.31 *As the set of static local axioms, $SLA_\Sigma(X,V)$, is a subset of $SA_\Sigma(X,V)$, the set of static global axiom, the definedness predicate of a static local axiom is the same as those of a static global axiom.*

As we said before, a Static Axiom is evaluated to a Change of Global State. This change is simply the identity when the Static Axiom is well defined and it is undefined otherwise.

Definition 4.32 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, $I \in Interp_X$ an interpretation function for variables, the evaluation of the Static (global or local) Axiom, $sa \in SA_\Sigma(X,V)$ is given by the partial function:*

$$[[\_]]_{SA,I} : SA_\Sigma(X,V) \to CST_A$$

defined by:

$$[[sa]]_{SA,I}(st) = \begin{cases} st, \text{ if } D_{SA,I,st}(sa) \\ \textit{undefined, otherwise} \end{cases}$$

The evaluation of a Static Axiom, sa, is a change of global state in CST_A. For each global state, st, this change of global state either returns the same global state st is the definedness predicate of the static axiom sa evaluates to true, or returns *undefined*.

Example 4.33 *Given the presentation* $PRES = (\Sigma, V, X, Ax)$ *of example 3.42, and the* Σ*-structure* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *of example 4.15, with the global state* st_4 *of example 4.23 and the interpretation* I *of variables of example 4.18, the definedness predicate of some static axioms is given by:*

$$
\begin{array}{lcl}
D_{SA,I,st_4}(Bqueue = emptyqueue\_bool) & \Leftrightarrow & D_{T,I,st_4}(Bqueue) \wedge D_{T,I,st_4}(emptyqueue\_bool) \\
 & & \wedge[[st_4, Bqueue]]_{T,I} = [[st_4, emptyqueue\_bool]]_{T,I} \\
 & \Leftrightarrow & st_4(Bqueue) = EMPTYQUEUE\_B \\
 & \Leftrightarrow & [] = [] \\
D_{SA,I,st_4}(//(put\_nat(zero), put\_nat(y)) = & & \\
\qquad //(get\_nat(zero), get\_nat(y))) & \Leftrightarrow & (([0,1],[]), put_{0,1}, ([1,0,0,1],[])) \in TRS_\Sigma \ s.t. \\
 & & [[st_4, //(put\_nat(zero), put\_nat(y)]]_{T,I} = put_{0,1}, \wedge \\
 & & (([0,1],[]), get_{0,1}, ([],[])) \in TRS_\Sigma \ s.t. \\
 & & [[st_4, //(get\_nat(zero), get\_nat(y)]]_{T,I} = get_{0,1} \wedge \\
 & & ([1,0,0,1],[]) \neq ([],[]) \\
D_{SA,I,st_4}(\neg D(pred(zero))) & \Leftrightarrow & \neg D_{T,I,st_4}(pred(zero)) \\
 & \Leftrightarrow & [[st_4, pred(zero)]]_{T,I} \ doesn't \ exist \\
 & \Leftrightarrow & PRED(0) \ doesn't \ exist
\end{array}
$$

The definedness predicate $D_{SA,I,st_4}(Bqueue = emptyqueue\_bool)$ is true because the local state $Bqueue$ is interpreted to $[]$ as it is for the $emptyqueue\_bool$ term.

The definedness predicate $D_{SA,I,st_4}(//(put\_nat(zero), put\_nat(y)) = //(get\_nat(zero), get\_nat(y))$ is false because the only two triples, whose transition is the evaluation of $//(put\_nat(zero), put\_nat(y))$ and $//(get\_nat(zero), get\_nat(y))$ respectively, have the same initial global state $([0,1],[])$ (i.e. the global state st_4), but not the same final global state $(([1,0,0,1],[]) \neq ([],[]))$.

The definedness predicate $D_{SA,I,st_4}(\neg D(pred(zero)))$ is true because the operation $PRED$ in $A_\Sigma = (A, F_A)$ is not defined over the natural 0.

Example 4.34 *Given the presentation* $PRES = (\Sigma, V, X, Ax)$ *of example 3.42, and the* Σ*-structure* $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ *of example 4.15, with the global state* st_4 *of example 4.23 and the interpretation* I *of variables of example 4.18, the evaluation of the static axioms of example 4.33 is given by:*

$$[[Bqueue = emptyqueue\_bool]]_{SA,I}(st_4) = st_4$$
$$[[//(put\_nat(zero), put\_nat(y)) = //(get\_nat(zero), get\_nat(y))]]_{SA,I}(st_4) \ undefined$$
$$[[\neg D(pred(zero))]]_{SA,I}(st_4) = st_4$$

The Static Axiom $Bqueue = emptyqueue\_bool$ is evaluated to a Change of Global State, that changes the global state st_4 to the same global state st_4, because the definedness predicate associated to this Static Axiom and with this global state is well defined. The other global states are changed to themselves, with this Change of Global State, if the corresponding definedness predicate is well defined, they are changed to "undefined" otherwise.

The Static Axiom $//(put\_nat(zero), put\_nat(y)) = //(get\_nat(zero), get\_nat(y))$ is evaluated to a Change of Global State, that changes the global state st_4 to "undefined", as the definedness predicate has evaluated to false.

The Static Axiom $\neg D(pred(zero))$ is evaluated to a Change of Global State, that changes the global state st_4 to the same global state st_4.

See example 4.33 for the associated definedness predicates.

4.7 Evaluation of Dynamic Axioms

A Dynamic Axiom is evaluated to a Change of Global State. For a given current state, the new state is obtained by applying the precondition of the Dynamic Axiom on the current state, and then by applying the postcondition of the Axiom to the resulting state. An smf changes a given state by the sma appearing in it, a local state is then changed to the value associated to the term affected to the local state. If a Static Local Axiom sa appears in a smf it either doesn't change the current global state if sa is well defined, or let's the global state become undefined.

Definition 4.35 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, $I \in Interp_X$ an interpretation function for variables, the evaluation of State Modification Atoms $sma \in SMA_\Sigma(X, V)$ is given by the partial function:*

$$[[\_]]_{SMA,I} : SMA_\Sigma(X, V) \to CST_A$$

defined by:

$$[[v := t]]_{SMA,I}(st)(v') = \begin{cases} st(v'), \; if\, v' \neq v \\ [[st, t]]_{T,I}, \; if\, v' = v \; and\, [[st, t]]_{T,I} \; exists \\ undefined, \; otherwise \end{cases}$$

A State Modification Formula $v := t$ is evaluated to a Change of Global State that returns for each current Global State. st, the same Global State except for the local state v that takes the evaluation of t, or an undefined Global State if the term t cannot be evaluated.

Definition 4.36 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, $I \in Interp_X$ an interpretation function for variables, the evaluation of State Modification Formula $smf \in SMF_\Sigma(X, V)$ is given by the partial function:*

$$[[\_]]_{SMF,I} : SMF_\Sigma(X, V) \to CST_A$$

defined by:

1. $[[\varepsilon]]_{SMF,I} = id_{V_A}$
2. $[[v := t]]_{SMF,I} = [[v := t]]_{SMA,I}$
3. $[[sla]]_{SMF,I} = [[sla]]_{SA,I}$
4. $[[f \& g]]_{SMF,I} = [[g]]_{SMF,I} \circ [[f]]_{SMF,I}$
5. $[[\exists.x(f)]]_{SMF,I} = [[f]]_{SMF,I'}, I' \in Interp_{X,I,x}$

The evaluation of the empty smf, ε, is the identity, i.e each global state remains unchanged by the change of global state id_{V_A}.

The evaluation of a State Modification Atom, $v := t$, is the Change of Global State changing only the specified local state, v, appearing in the sma and this local state takes the evaluation of the term t appearing in the sma.

The evaluation of a Static local Axiom is the Change of Global State that returns the same Global State if the Static local Axiom is well defined for the given local state, and undefined otherwise.

The evaluation of a State Modification Formula, resulting of the concatenation of two other State Modification Formula, is the Change of Global State resulting of the composition of the Change of Global State of the second smf with the Change of Global State of the first smf. This means that when a new Global State is obtained by the application of a composition $f \& g$ of smf, the Global State is firstly changed by the first smf, f, of the composition, and the resulting Global State is then changed by the second smf, g, of the composition.

The evaluation of a State Modification Formula overalled by the $\exists x.$ quantifier is the evaluation of the same smf with an interpretation of variables that changes the x variable only. This interpretation is *one* between all the possible interpretations.

Definition 4.37 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, $I \in Interp_X$ an interpretation function for variables, the evaluation of a Dynamic Axiom $da \in DA_\Sigma(X, V)$ is given by the partial function:*

$$[[\_]]_{DA,I} : DA_\Sigma(X, V) \to CST_A$$

defined by:

$$[[evnt : f \to g]]_{DA,I} = [[g]]_{SMF,I} \circ [[f]]_{SMF,I}$$

The evaluation of a Dynamic Axiom is the Change of global State given by the composition of the Change of Global State associated to the postcondition, g, of the Dynamic Axiom with the Change of Global State associated to the precondition, f, of the Dynamic Axiom.

Definition 4.38 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ a Σ-structure, st a Global State, $da \in DA_\Sigma(X, V)$ a Dynamic Axiom, the definedness predicate of da, $D_{DA,I,st}(da)$, is given by:*

$$D_{DA,I,st}(da) \Leftrightarrow [[da]]_{DA,I}(st) exists$$

Example 4.39 *Given the presentation $PRES = (\Sigma, V, X, Ax)$ of example 3.42, and the Σ-structure $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ of example 4.15, with the global state st_4 of example 4.23 and the interpretation I of variables of example 4.18, the evaluations of some Dynamic Axioms is given by:*

$$\begin{aligned}
[[get_nat(m):\ & \neg(Nqueue = emptyqueue_nat) \rightarrow \\
& (m = first_nat(Nqueue))\&Nqueue := remove_nat(Nqueue)]]_{DA,I} \\
& = [[(m = first_nat(Nqueue))\&Nqueue := remove_nat(Nqueue)]]_{SMF,I} \circ \\
& \qquad [[\neg(Nqueue = emptyqueue_nat)]]_{SMF,I}
\end{aligned}$$

Let us see now, how this Change of Global State changes the states, st_1, st_2 of example 4.23.

$$\begin{aligned}
& [[(m = first_nat(Nqueue))\&Nqueue := remove_nat(Nqueue)]]_{SMF,I} \circ \\
& \quad [[\neg(Nqueue = emptyqueue_nat)]]_{SMF,I}(st_1) \\
& \qquad = [[(m = first_nat(Nqueue))\&Nqueue := remove_nat(Nqueue)]]_{SMF,I}(st_1) \\
& \qquad = [[Nqueue := remove_nat(Nqueue)]]_{SMF,I} \circ [[m = first_nat(Nqueue)]]_{SLA,I}(st_1) \\
& \qquad = [[Nqueue := remove_nat(Nqueue)]]_{SMA,I}(st_1) \\
& \qquad = ([[st_1, remove_nat(Nqueue)]]_{T,I}, [[st_1, BNqueue]]_{T,I}) \\
& \qquad = ([1, 2, 0], [T, F, F, T]) \\
& [[(m = first_nat(Nqueue))\&Nqueue := remove_nat(Nqueue)]]_{SMF,I} \circ \\
& \quad [[\neg(Nqueue = emptyqueue_nat)]]_{SMF,I}(st_2) \\
& \qquad = \ undefined
\end{aligned}$$

The evaluation of this Dynamic Axiom is a Change of Global State, that changes the global state st_2 to "undefined", since the $Nqueue$ local state is an empty queue and the precondition of this Dynamic Axiom states that this local state must be different from the empty queue.

The global state st_1 is changed to a new global state, where the $Bqueue$ local state remains unchanged and where the $Nqueue$ local state is decreased of its first element, the natural number 1.

Properties 4.40 *Given $evnt \in (T_\Sigma(X, V))_{(ev \in EV)}$ an event, $f, g \in SMF_\Sigma(X, V)$, we have the following properties for the evaluation of dynamic axioms:*

$$[[evnt : \varepsilon \rightarrow f \ \&\ g]]_{DA,I} = [[evnt : f \rightarrow g]]_{DA,I} = [[evnt : f \ \&\ g \rightarrow \varepsilon]]_{DA,I}$$

4.8 Model

Definition 4.41 *Let $PRES = (\Sigma, V, X, Ax)$ be a Presentation, a Σ-model, Mod_Σ, is a Σ-structure, $Mod_\Sigma = (A_\Sigma, TRS_\Sigma)$ such that:*

1. $\forall sa \in Ax \cap SA_\Sigma(X, V) then$
   $\forall I \in Interp_X, \forall st \in V_A \Rightarrow D_{SA,I,st}(sa)$
2. $\forall (a, w, b) \in TRS_\Sigma$ *then*
   $\exists I \in Interp_X, \exists da \in dDA_\Sigma(X, V)$ *such that*
   - *a.* $da = evnt : f \rightarrow g$
   - *b.* $w = [[a, evnt]]_{T,I}$
   - *c.* $D_{DA,I,st}(da)$
   - *d.* $b = [[da]]_{DA,I}(a)$

A Σ-Structure is a model if all Global Static Axioms are well defined for every interpretation of variables and for every global state, i.e. Global Static Axioms have to be true in all cases.

A Σ-Structure is a model if for every transition in TRS_Σ, there exists an interpretation of variables and a derived Dynamic Axiom, whose event evaluates to the transition and the final state of the transition is reached when the Dynamic Axiom is applied on the initial state of the transition. We can say that a Σ-Structure is a model if for every transition in TRS_Σ, there is a derived Dynamic Axiom that "explains" how the final global state of the transition is reached from the initial global state.

Remark 4.42 *Dynamic Axioms may have free variables, variables that do not have quantifier, in this case the default quantifier is said to be $\exists$ quantifier. This appears in the condition 2. for Σ-models, by $\exists I \in Interp_X$. A Dynamic Axiom with free variables behaves like the same Dynamic Axiom with an overall $\exists$ quantifier, one for each free variable.*

If a free variable appears in a Global Static Axiom, the free occurrences of the same variable will be evaluated to the same value, and the Global Static Axiom must be true for all possible values of all free variables. A Global Static Axiom with free variables behaves like the same Global Static Axiom with an overall $\forall$ quantifier, one for each free variable. This appears in the condition 1. for Σ-models, by $\forall I \in Interp_X$.

Two occurrences of the same variable appearing in a Global Static Axiom, each with its own quantifier will be evaluated to different values as if they were different variables.

Example 4.43 *Given the presentation $PRES = (\Sigma, V, X, Ax)$ of example 3.42, and the Σ-structure $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$ of example 4.15, let us see if this Σ-structure is a model of the presentation.*

The Static Axioms, except $//(ev_1, ev_2) = //(ev_2, ev_1)$ and $//(ev_1, //(ev_2, ev_3)) = //(//(ev_1, ev_2), ev_3)$ of the given presentation are satisfied for whatever global state we consider and for whatever value is given to the variables. The chosen operations of the Σ-structure are exactly those we want to have the naturals, booleans, queues of naturals and booleans.

The Static Axiom $//(ev_1, ev_2) = //(ev_2, ev_1)$ is satisfied if the two parts of the equality evaluate for each global state and for each interpretation of the variables to the same event in A_{ev} or if they evaluate to two different events that are involved in the same transitions of TRS_Σ. The first case, evaluation to the same event, is not true, because the definition of the $//$ operator in F_A, is chosen to give different events. The other possibility, same behavior, is not true because if $I(ev_1) = put_0$ and $I(ev_2) = put_1$ then $//(ev_1, ev_2)$ evaluates to $put_{0,1}$ while $//(ev_2, ev_1)$ evaluates to $put_{1,0}$, and the set of transitions TRS_Σcontains two triples:

$$(([2,3],[]), put_{1,0}, ([0,1,2,3],[])), (([2,3],[]), put_{0,1}, ([1,0,2,3],[]))$$

with the same initial global state $([2,3],[])$, whose transitions are $put_{1,0}$ and $put_{0,1}$ respectively, but whose final global states are different.

The static axiom $//(ev_1, //(ev_2, ev_3)) = //(//(ev_1, ev_2), ev_3)$ is satisfied because the definition of the $//$ operator in F_A possesses this property. For example, if $I(ev_1) = put_0$, $I(ev_2) = put_1$ and $I(ev_3) = put_2$, then $//(ev_2, ev_3)$ evaluates to $put_{1,2}$ and $//(ev_1, //(ev_2, ev_3))$ to $put_{0,1,2}$, while $//(ev_1, ev_2)$ evaluates to $put_{0,1}$ and $//(//(ev_1, ev_2), ev_3)$ evaluates to $put_{0,1,2}$.

This Σ-structure is not a model because the Static Axiom $//(ev_1, ev_2) = //(ev_2, ev_1)$ is not satisfied.

Consider now the same Σ-structure, where this Static Axiom has been taken off. All Static Axiom are satisfied now. It rests to see if all the transitions in TRS_Σ can be associated to a Dynamic Axiom in $PRES = (\Sigma, V, X, Ax)$ that explains their behavior. The given Σ-structure contains several transitions in TRS_Σ. We see that the first one has the associated derived Dynamic Axiom whose event is $//(put\_nat(zero), put\_nat(succ(zero)))$. The calculus of the derived Dynamic Axiom is let to the reader. This evaluation of this Dynamic Axiom applied to the initial global state: $([1,2],[T,F,T)$ returns the final global state $([1,0,1,2],[T,F,T])$. The transition $(([1,2],[T,F,T]), put_{0,1}, ([1,0,1,2],[T,F,T]))$ is then explained with a Dynamic Axiom of the presentation. We see that the other transitions of TRS_Σ are also explained with Dynamic Axioms of $Struct_\Sigma = (A_\Sigma, TRS_\Sigma)$.

This new Σ-structure is then a model according to definition 4.41.

Remember that meta-rules are used to enlarge the set of dynamic axioms. Given a presentation with dynamic axioms and meta-rules, we can derive a new presentation, called *derived* presentation, with no meta-rules and with an enlarged set of derived dynamic axioms. The current semantics, explained above, is associated to the derived presentation. We assert, without prooving it, that the semantics given to the derived presentation is reachable from the semantics we can give to the previous presentation considering only the restricted set of dynamic axioms.

Conjecture 4.44 *Given $PRES = (\Sigma, V, X, Ax)$ a presentation, there exists a function $Meta'$ that causes the following diagram to commute.*

$$\begin{tikzcd} PRES \arrow[r, "\mathrm{Meta}"] \arrow[d, "\mathrm{Mod}"'] & PRES' \arrow[d, "\mathrm{Mod}"] \\ \{Models\} \arrow[r, "\mathrm{Meta}'"] & \{Models'\} \end{tikzcd}$$

where $PRES'$ is the derived presentation, (definition 4.11), Meta is an application that furnishes the derived presentation of a given presentation: $\mathrm{Meta}(PRES) = PRES'$. *The function* Mod *returns all the models of given presentation. The sets $\{Models\}$ and $\{Models'\}$ are the sets of all Σ-structures that are models of $PRES$, $PRES'$ respectively, when no meta-rules are considered. (This is the case for $PRES$ only, as in $PRES'$ all meta-rules have been removed)*

This conjecture is useful for computation purposes. The derived presentation $PRES'$ contains a possibly infinite set of dynamic axioms. In a computation process it would be impossible to compute all these dynamic axioms. If the conjecture is true, the computation process could work on $PRES$, which is made of a finite set of dynamic axioms and a finite set of meta-rules.

Chapter 5

Example: Arc Extensions for Coloured Petri Nets in Access

Access extends several formal languages [DiMa94], as Algebraic Petri Nets (PN), Coloured Petri Nets (CPN), Gamma, ACP, CO-OPN.

This chapter is dedicated to the way how Access generalizes Arc Extensions for Coloured Petri Nets.

This chapter firstly presents how "classical" Petri Nets are described with Access. It then remembers the formal definitions of CPN and Arc Extensions for CPN. It then continues with the description in Access of some examples of CPN and Arc Extensions. Based on these examples, a general method to derive Access presentations from CPN with arc extensions is given. This chapter ends with a discussion about the differences between CPN specifications and Access presentations, and gives a conjecture about a possible isomorphism between Petri Nets models and Access presentations models.

5.1 Petri Nets in Access

Petri Nets are naturally represented in Access. Each place p of a PN becomes a Local State (definition 3.14) in the Access system, each transition t becomes an Event. The algebraic specification is the integer, it is the sort of each local state. The value of a local state is the number of tokens of the corresponding place. Each transition, t, is defined with a dynamic axiom. Input (p, t) and output (t, p) arcs from a place p to the transition t or vice-versa, become SMA, State Modification Atom in the dynamic axiom.

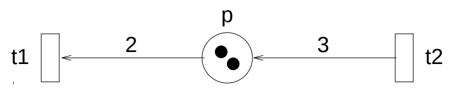

Figure 5.1: Trivial PN

Example 5.1 *Given the trivial PN of figure 5.1, the corresponding* ACCESS *presentation is given by* $PRES = (\Sigma, V, X, Ax)$ *where:*

$$
\begin{array}{lll}
1. \ \Sigma & = & (S, EV, F) \textit{ with:} \\
 & & \quad \begin{array}{lll} S & = & \{int\} \\ EV & = & \{ev\} \\ F & = & \{ \textit{ Operations over the sort int} \\ & & \quad \begin{array}{lll} 0: & & \rightarrow int \\ +: & int\ int & \rightarrow int \\ -: & int\ int & \rightarrow int \end{array} \\ & & \textit{Events} \\ & & \quad \begin{array}{ll} t1: & \rightarrow ev \\ t2: & \rightarrow ev \end{array} \\ & & \textit{Operations over events} \\ & & // : ev\ ev \rightarrow ev\} \end{array} \\
 & & \textit{with } \subseteq_S = \emptyset,\ \subseteq_{EV} = \emptyset \\
2. \ V & = & V_{int} = \{p\} \\
3. \ X & = & \{x, y, \ldots, \overline{ev}_1, \overline{ev}_2, \overline{f}_1, \overline{f}_2, \overline{g}_1, \overline{g}_2\} \\
4. \ Ax & = & \{ \textit{ Static Global Axioms over int:} \\
 & & \quad 0 + x = x \\
 & & \quad x - 0 = x \\
 & & \quad \neg(D(0 - x)) \\
 & & \quad \ldots \\
 & & \quad \textit{Dynamic axioms:} \\
 & & \quad \begin{array}{lll} t1: & p := p - 2 & \rightarrow \varepsilon \\ t2: & \varepsilon & \rightarrow p := p + 3 \end{array} \\
 & & \quad \textit{Meta-rules:} \overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \quad \rightsquigarrow \quad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2\}
\end{array}
$$

The signature, Σ, contains the algebraic specification for the integers (the sort int, with the operations $+, -, 0$); the sort ev for events; two operation producing events, $t1, t2$, corresponding to the transitions of the given PN, and the // operation over events.

The set V of local states contains the place p. The set X of variables contains all variables used in the Axioms of the Ax set.

The set Ax of Axioms is made of Static Global Axioms defining the sort int; Dynamic Axioms, one for each transition ti; and of Meta-rules.

The Dynamic Axiom for $t1$ states that the $t1$ event decreases the local state p from the value 2 and requires no further action, this correspond in the given PN, to the arc $(p, t1)$ labelled with 2 tokens. Similarly, the Dynamic Axiom related to the $t2$ event requires no previous action and increases p with the value 3, as does the arc $(t2, p)$ in the PN.

The Meta-rule gives the behavior of the // operator over the events as the "true" parallelism. This behavior has been already explained in example 3.36.

5.2 Basic Definitions

5.2.1 CPN

The representation, with Petri Nets, of systems, which contain several parts similar, but not identical, requires to represent several times the same subnet, one for each of these parts. Large systems become difficult to represent in that way. CPN are an answer to that problem. A given place in a CPN has an associated type c and its marking consists of a multi-set of this type. Each token of the place has a value of type c. The different colours are given by the different values; they represent different places in a classical Petri Net.

Arcs are no more labelled with integers, but with expressions concerning multi-sets. Each arc has then an arc expression attached, each value taken by the arc expression represents a different arc in a classical PN.

Each Transition has a boolean expression attached, a *Guard*, a step is enabled in the CPN, if the bindings of the variables appearing in the Guard of the transition and on the arc expression of adjacent arcs fulfill the Guard expression.

Finally a step is enabled if the occurring bindings fulfill the Guards of the transitions involved in the step and if the bindings together with the current marking allow the transitions to be fired.

Definition 5.2 *[Jens92] A CP-net is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ such that:*

1. Σ *is a finite set of non-empty types, called* **colour sets**
2. P *is a finite set of* **places**
3. T *is a finite set of* **transitions**
4. A *is a finite set of* **arcs** *such that*
   $P \cap T = P \cap A = T \cap A = \emptyset$
5. N *is a* **node** *function. N is defined from A into $P \times T \cup T \times P$.*
6. C *is a* **colour** *function. C is defined from P into Σ.*
7. G *is a* **guard** *function. G is defined from T into expressions such that:*
   $\forall t \in T : [Type(G(t)) = \mathcal{B} \wedge Type(Var(G(t))) \subseteq \Sigma]$
8. E *is an* **arc expression** *function. E is defined from A into expressions such that:*
   $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$
   where $p(a)$ is the place of $N(a)$.
9. I *is an* **initialization** *function. I is defined from P into closed expressions such that:*
   $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

Where $\mathcal{B}$ is the the boolean type; $Type(expr)$ denotes the type of the expression expr; $Var(expr)$ denotes the set of variables of expression expr; and $Type(Var(expr))$ denotes the set of types $\{Type(v) | v \in Var(expr\}$, and c_{MS} stands for the multiset whose elements are of type c.

The node function N defines the input and output arcs.

The guard function G associates to each transition t a boolean expression $expr_t$, while the arc expression function E associates to each arc a an expression $expr_a$.

Definition 5.3 *[Jens92] A step Y of a CPN is* enabled *in a marking M iff:*

$$\forall p \in P : \sum_{(t,b) \in Y} E(p,t) < b > \leq M(p)$$

A step is enabled if for each place p, the binding b and the marking M ensure that tokens can be removed by the input arcs adjacent to each transition involved in the step.

When a step Y is enabled, the new marking M' is given by:

$$\forall p \in P : M'(p) = M(p) - \sum_{(t,b) \in Y} E(p,t) < b > + \sum_{(t,b) \in Y} E(t,p) < b >$$

The new marking is given by the current marking where the removed tokens are removed before the added tokens are added. The removed (or added) tokens are given by the arc expressions attached to the input (or output) arcs adjacent to the transitions t involved in the step Y.

5.2.2 Arc Extensions for CPN

A CP-net with test arcs and inhibitor arcs, is a CP-net with usual input, output arcs and with two other sorts of arcs: the test arcs and the inhibitors arcs. These last two sorts of arcs are useful for testing the values of places without changing these values.

Each arc has an arc expression attached. For an input arc, this arc expression represents the tokens to remove. For an output arc, the arc expression stands for the tokens to add. For a test arc, it stands for the minimal value that the adjacent place can take. For an inhibitor arcs, it stands for the maximal value that the adjacent place can take.

Definition 5.4 *[Jens94] A CP-Net with test arcs and (generalized) inhibitor arcs is a tuple* $CPN_{TI} = (CPN, TS, IS)$ *where:*

1. $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ *is a CP-net, see definition 5.2*
2. $TS = (A_T, N_T, E_T)$ *is a test arc specification with:*
   a. A_T *is a set of test arcs, such that:* $A_T \cap (P \cup T \cup A) = \emptyset$.
   b. $N_T : A_T \rightarrow P \times T$ *is a test node function defined on* A_T.
   c. E_T *is the test expression function defined from* A_T *into expressions such that:*
   $$\forall a \in A_T : [Type(E_T(a)) = C(p(a))_{MS} \wedge Type(Var(E_T(a))) \subseteq \Sigma]$$
   where $p(a)$ *is the place of of* $N_T(a)$
3. $IS = (A_I, N_I, E_I)$ *is a inhibitor arc specification with:*
   a. A_I *is a set of inhibitor arcs, such that:* $A_I \cap (P \cup T \cup A \cup A_T) = \emptyset$.
   b. $N_I : A_I \rightarrow P \times T$ *is a inhibitor node function defined on* A_I.
   c. E_I *is the inhibitor expression function defined from* A_I *into expressions such that:*
   $$\forall a \in A_I : [Type(E_I(a)) = C(p(a))_{MS} \wedge Type(Var(E_I(a))) \subseteq \Sigma]$$
   where $p(a)$ *is the place of of* $N_I(a)$

CPN with arc extensions are CPN with two additional sorts of arcs, test and inhibitor arcs. Each arc of the CPN has arc expression attached. These expressions must be of the same type of the type of the place to which they are adjacent and the type of their variables must be part of the authorized types Σ.

Definition 5.5 *[Jens94] A step* Y *of a* CPN_{TI} *is enabled in a marking* M *iff:*

$1.\forall p \in P : \sum_{(t,b)\in Y} E(p,t) < b > \leq M(p)$
$2.\forall a \in A_T : \forall (t,b) \in Y : E_T(a) < b > \leq M(p(a)) - \sum_{(t',b')\in Y} E(p(a),t') < b' >$
$3.\forall a \in A_I : \forall (t,b) \in Y : E_I(a) < b > \geq M(p(a)) + \sum_{(t',b')\in Y} E(t',p(a)) < b' >$
$p(a)$ *is the place or arc* a.

A step is enabled if the input arcs involved in the step can all be fired, and if the expression attached to each test (inhibitor) arc is less (more) than the value of the adjacent place p previously decreased (increased) from all arc expressions of input (output) arcs involved in the step.

5.3 CPN with Arc Extensions in Access

The colours used in the CPN are represented by algebraic specifications in Access. Each transition of the CPN becomes an event of the Access system. The boolean expression attached to a transition becomes a *SLA* (Static Local Axioms, see definition 3.23) in the dynamic axiom describing the transition.

The arcs adjacent to a transition become *SMF* (State Modification Formula, see definition 3.26) of the dynamic axiom describing the transition: input, output arcs of the CPN become *SMA* (State Modification Atoms, see definition 3.25); test and inhibitor arcs become *SLA* (Static Local Axioms, see definition 3.23).

This section gives some simple examples of CPN and their presentation in Access, and ends with the generalization of CPN specification into Access presentations.

5.3.1 Examples

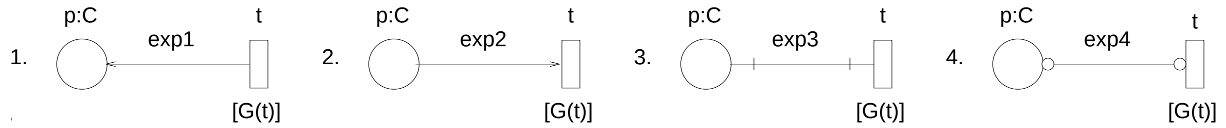

Figure 5.2: Basic arc extensions

Example 5.6 *Figure 5.2 represents transitions with input, output, test and inhibitor arcs respectively. This basic examples of transitions are represented in* ACCESS, *with the following Dynamic Axioms:*

$$\begin{array}{lll} 1. & t : G(t) & \rightarrow p := p + exp1 \\ 2. & t : G(t) & \rightarrow p := p - exp2 \\ 3. & t : G(t) \,\&\, (p \geq exp3 = true) & \rightarrow \varepsilon \\ 4. & t : G(t) \,\&\, (p \leq exp4 = true) & \rightarrow \varepsilon \end{array}$$

The boolean expression $G(t)$ has to be satisfied to enable the transition t. In ACCESS *this is expressed by the SLA: $G(t)$, present in the dynamic axiom describing t.*

The output arc needs no constraints and adds $exp1$ to the place p.

The input arc needs the $-$ (minus) function to succeed and then it subtracts $exp2$ to the place p. There is no constraint in the second case, because the $-$ operation over the colour C is not defined, if $exp2 \geq p$. That is to say, that a predicate (see definition 3.21) is associated to the $-$ function as a Static Global Axiom.

The test arc needs the constraint $p \geq exp3$ to enable transition t, this arc has no effect on the value of p.

The inhibitor arc tests if $p \leq exp3$ and doesn't affect p.

The $+$ and $-$ operations are the $+, -$ operation over multisets of type C.

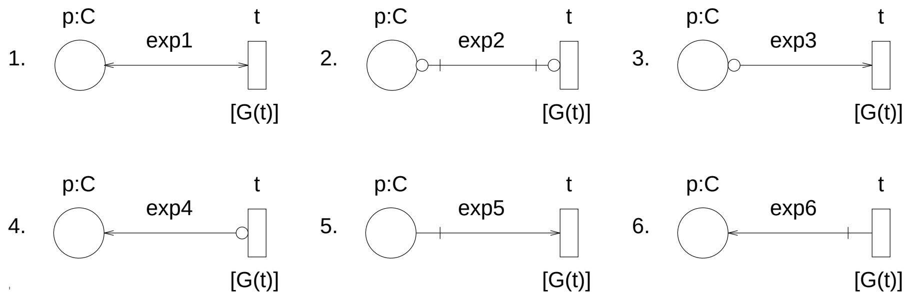

Figure 5.3: Compound arcs

Example 5.7 *Figure 5.3 represents transitions with combinations of input, output, test and inhibitor arcs, the arcs between the place p and the transition t have the same expression. For example the combination of arcs in 1. is made of an input arc and an output arc having each the same arc inscription $exp1$.*

The semantics of these compound arcs is expressed in ACCESS*with the following dynamic axioms.*

$$\begin{array}{lll} 1. & t : G(t) \,\&\, (p \geq exp1 = true) & \rightarrow \varepsilon \\ 2. & t : G(t) \,\&\, p = exp2 & \rightarrow \varepsilon \\ 3. & t : G(t) \,\&\, p = exp3 & \rightarrow p := \emptyset \\ 4. & t : G(t) \,\&\, p = \emptyset & \rightarrow p := exp4 \\ 5. & t : G(t) \,\&\, (p \geq exp5 + exp5 = true) & \rightarrow p := p - exp5 \\ 6. & t : G(t) \,\&\, (p \geq exp6 = true) & \rightarrow p := p + exp6 \end{array}$$

These dynamic axioms, one for each CPN mean that:

1. *The* **Reserve** *arc ensures that the place p is greater than the expression $exp1$. It doesn't affect p.*

2. *The* **Equal** *arc doesn't affect p, but ensures that $p = exp2$.*

3. *The* **Clear** *arc removes any value from the place p by setting p to the emptyset, this is done if the value of the place is equal to $exp3$.*

4. *The* **Set** *arc changes the value of p to $exp4$, if place p is empty.*

5. *The* **Halve** *arc takes away $exp5$ from p if p is over $2 \star exp5$.*

6. *The* **Double** *arc add $exp6$ to p is p is over $exp6$.*

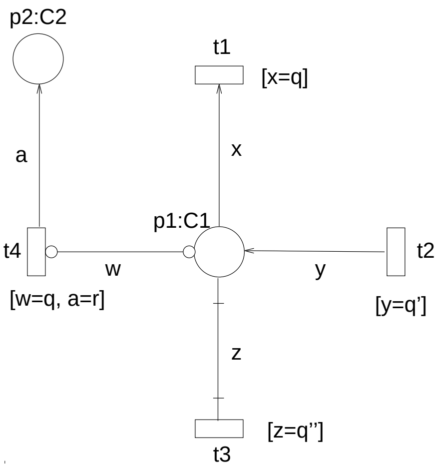

Figure 5.4: Simple CPN with several arcs

Example 5.8 *Figure 5.4 represents a CPN with 2 places, $p1, p2$ of color type $C1, C2$ respectively, and with 4 transitions, $t1, \ldots, t4$ each with a guard associated. The place $p1$ is connected to the four transitions by the mean of 4 different types of arcs, each with its arc expression.*

Its complete description in ACCESS, *is given by the presentation* $PRES = (\Sigma, V, X, Ax)$ *where:*

1. $\Sigma = (S, EV, F)$ *with:*

$$
\begin{array}{lll}
S & = & \{bool, C1, C2, C1_{MS}, C2_{MS}\} \\
EV & = & \{ev\} \\
F & = & \{ \text{ Operations over bool:} \\
 & & \begin{array}{lll}
true: & & \to bool, \\
false: & & \to bool, \\
\neg: & bool & \to bool, \\
\vee: & bool & \to bool, \\
\wedge: & bool & \to bool, \\
\multicolumn{3}{l}{\text{Operations over } C1, C2\text{:}} \\
0_{C1}: & C1 & \to C1, \\
+_{C1}: & C1 & \to C1, \\
-_{C1}: & C1 & \to C1, \\
0_{C2}: & C1 & \to C1, \\
+_{C2}: & C2 & \to C2, \\
-_{C2}: & C2 & \to C2, \\
\multicolumn{3}{l}{\text{Operations over } C1_{MS}, C2_{MS}\text{:}} \\
q, q', q'': & & \to C1_{MS}, \\
r: & & \to C2_{MS}, \\
\emptyset_{C1_{MS}}: & & \to C1_{MS}, \\
+_{C1_{MS}}: & C1_{MS} & \to C1_{MS}, \\
-_{C1_{MS}}: & C1_{MS} & \to C1_{MS}, \\
MAKE_{C1_{MS}}: & C1 & \to C1_{MS}, \\
\geq_{C1_{MS}}: & C1_{MS}\, C1_{MS} & \to bool, \\
\emptyset_{C2_{MS}}: & & \to C1_{MS}, \\
+_{C2_{MS}}: & C2_{MS} & \to C2_{MS}, \\
-_{C2_{MS}}: & C2_{MS} & \to C2_{MS}, \\
MAKE_{C2_{MS}}: & C2 & \to C2_{MS}, \\
\geq_{C2_{MS}}: & C2_{MS}\, C2_{MS} & \to bool \\
\multicolumn{3}{l}{\text{Transitions as events:}} \\
t1: & C1 & \to ev, \\
t2: & C1 & \to ev, \\
t3: & C1 & \to ev, \\
t4: & C1\, C2 & \to ev \\
\multicolumn{3}{l}{\text{Operations over events:}} \\
//: & ev\, ev & \to ev
\end{array}
\end{array}
$$

with $\subseteq_S = \emptyset$, $\subseteq_{EV} = \emptyset$

2. $V = V_{C1} \cup V_{C2} \cup V_{C1_{MS}} \cup V_{C2_{MS}}$ *where:*

$$
\begin{array}{ll}
V_{C1} = V_{C2} & = \emptyset \\
V_{C1_{MS}} & = \{p1\} \\
V_{C2_{MS}} & = \{p2\}
\end{array}
$$

3. $X = \{x, y, z, w, a, \ldots\}$

4. $Ax \;=\; \{$ *Static Global Axioms over the sorts S:*

...

Dynamic Axioms:

$$
\begin{array}{lll}
t1(x): & (x = q) & \rightarrow p1 := p1 -_{C1_{MS}} x \\
t2(y): & (y = q') & \rightarrow p1 := p1 +_{C1_{MS}} y \\
t3(z): & (z = q'')\ \&\ (p1 \geq z) & \rightarrow \varepsilon \\
t4(w,a): & (w = q \wedge a = r)\ \&\ (p1 \leq w) & \rightarrow p2 := p2 +_{C2_{MS}} a
\end{array}
$$

$$
\begin{array}{ll}
//(t1(x), t2(y)): & (x = q)\ \&\ (y = q') \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x)\ \&\ (p1 := p1 +_{C1_{MS}} y) \\
//(t1(x), t3(z)): & (x = q)\ \&\ (z = q'')\ \&\ ((p1 -_{C1_{MS}} x) \geq z) \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x) \\
//(t1(x), t4(w,a)): & (x = q)\ \&\ (w = q \wedge a = r)\ \&\ (p1 \leq w) \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x)\ \&\ (p2 := p2 +_{C2_{MS}} a) \\
//(t2(x), t3(z)): & (y = q')\ \&\ (z = q'')\ \&\ (p1 \geq z) \rightarrow \\
& (p1 := p1 +_{C1_{MS}} y) \\
//(t2(x), t4(w,a)): & (y = q')\ \&\ (w = q \wedge a = r)\ \&\ ((p1 +_{C1_{MS}} y) \leq w) \rightarrow \\
& (p1 := p1 +_{C1_{MS}} y)\ \&\ (p2 := p2 +_{C2_{MS}} a) \\
//(t3(x), t4(w,a)): & (z = q'')\ \&\ (w = q \wedge a = r)\ \&\ (p1 \geq z)\ \&\ (p1 \leq (w)) \rightarrow \\
& (p2 := p2 +_{C2_{MS}} a) \\
//(t1(x), t2(y), t3(z)): & (x = q)\ \&\ (y = q')\ \&\ (z = q'')\ \&\ ((p1 -_{C1_{MS}} x) \geq z) \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x)\ \&\ (p1 := p1 +_{C1_{MS}} y) \\
//(t1(x), t2(y), t4(w,a)): & (x = q)\ \&\ (y = q')\ \&\ (w = q \wedge a = r)\ \&\ ((p1 +_{C1_{MS}} y) \leq w) \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x)\ \&\ (p1 := p1 +_{C1_{MS}} y)\ \& \\
& (p2 := p2 +_{C2_{MS}} a) \\
//(t2(y), t3(z), t4(w,a)): & (y = q')\ \&\ (z = q'')\ \&\ (w = q \wedge a = r)\ \& \\
& (p1 \geq z)\ \&\ ((p1 + y) \leq w) \rightarrow \\
& (p1 := p1 +_{C1_{MS}} y)\ \&\ (p2 := p2 +_{C2_{MS}} a) \\
//(t1(x), t2(y), t3(z), t4(w,a)): & (x = q)\ \&\ (y = q')\ \&\ (z = q'')\ \&\ (w = q \wedge a = r)\ \& \\
& ((p1 -_{C1_{MS}} x) \geq z)\ \&\ ((p1 +_{C1_{MS}} y) \leq w) \rightarrow \\
& (p1 := p1 -_{C1_{MS}} x)\ \&\ (p1 := p1 +_{C1_{MS}} y)\ \& \\
& (p2 := p2 +_{C2_{MS}} a)\}
\end{array}
$$

The signature Σ contains the sorts, $C1, C2$ and the multisets of these sorts, $C1_{MS}, C2_{MS}$. The event sort is ev, and the set of operations is made of operations over the sorts, events and operations over events. The sorts are not ordered.

The operations over the sorts defines 4 constants q, q', q'' of sort $C1_{MS}$ and r of sort $C2_{MS}$, corresponding to the constants appearing in the given CPN. There are also operations over the sorts C_i and over the multisets as $+_{Ci_{MS}}, -_{Ci_{MS}}, MAKE_{Ci_{MS}}, \emptyset_{Ci_{MS}}$. The events are the transitions of the given CPN, their parameters are of the sort of the variables needed by the transition. For example, the transition $t1$ needs the variable i of sort $C1_{MS}$, because it appears in its guard and in the arc expression of the adjacent input arc; while the transition $t4$ needs the variable w of type $C1_{MS}$ and the variable a of type $C2_{MS}$.

The set V of local state is made of the places, $p1$ of sort $C1_{MS}$ and $p2$ of sort $C2_{MS}$. The set X of variables contains all variables appearing in the axioms.

The set of axioms contains static global axioms and Dynamic axioms.

Static global axioms are concerned with usual operations over multisets, for this reason they are not mentioned here.

Dynamic axioms explain each possible single event (single transition), each possible pair, triple and quadruple of event.

The single events, $t1(x), t2(y)$ are connected to only input, output arcs. For this reason, their dynamic axiom contains as precondition (before the $\rightarrow$) only the guard and as postcondition (after the $\rightarrow$) the

State Modification Atom corresponding to the arc expression.

The single events, $t3(z), t4(w, a)$ are connected to test, inhibitor arcs, for this reason, the precondition of their dynamic axiom contains the guard and the test and inhibitor conditions required by the test or inhibitor arc, and the postcondition contains SMA decreasing or increasing the value of the places connected to the transition by input or output arcs respectively.

Note that in this example we have written $(p \geq exp)$ as a shorthand for $(p \geq exp = true)$.

For events occurring in $//$, the precondition contains the guard of each event, and the postcondition contains all State Modification Atom required by all events. If test and inhibitor arcs are adjacent to the transitions, involved in the event, then in the precondition, test and inhibitor conditions have to be added. To respect the enabling of the steps, see definition 5.5, the test and inhibitor conditions must be adapted depending on which input, output arcs are involved in the step.

For example, the event $//(t1(x), t3(z))$ requires the condition $(p1 - x) \geq z$ because $t3(x)$ is connected to a test arc with arc expression z and because $t1$ occurs simultaneously and is connected to an input arc with expression x. On the other hand, the event $//(t2(y), t3(z))$ has the simple test condition $p1 \geq z$, because none of the transitions involved in the event is connected to an input arc. It goes the same for events occurring with inhibitor conditions.

In this example, we have chosen to describe each event by its own dynamic axiom, in order to explicitly show the behavior of the events, especially compound events involving test and inhibitors conditions.

The next section introduces the generalization of the ACCESS presentation of Petri Nets with Arc Extensions. For this generalization, we use meta-rules to explain compound events.

5.3.2 Generalization

Definition 5.9 *(Automatic description of CPN arc extensions into* ACCESS*). Given a $CPN_{TI} = (CPN, TS, IS)$ with arc extensions as in definition 5.4, the corresponding* ACCESS *presentation is given by $PRES = (\Sigma', V, X, Ax)$ where:*

$$
\begin{array}{llll}
1. & \Sigma' & = & (S, EV, F) \textit{ with:} \\
 & & & \begin{array}{lll} S & = & \{bool\} \cup \Sigma \cup (\Sigma)_{MS} \\ EV & = & \{ev\} \\ F & = & \{\textit{operations over the sorts } S\} \cup \\ & & \{t : Type(Var(t)) \rightarrow ev | t \in T\} \cup \\ & & \{// : ev\ ev \rightarrow ev\} \end{array} \\
 & & & \textit{with } \subseteq_S = \emptyset,\ \subseteq_{EV} = \emptyset \\
2. & V & = & \cup_{s \in \Sigma} V_{s_{MS}} \textit{ where:} \\
 & & & \quad V_{s_{MS}} = \{v \in P | C(v) = s\} \\
3. & X & = & \{\textit{All variables appearing in } Ax\}
\end{array}
$$

$$
\begin{array}{lll}
4.\quad Ax & = & \{\textit{Static Global Axioms over the sorts } S\ (\ldots)\} \cup \\
& & \{\textit{Dynamic axioms over single transitions} \\
& & \;t(Var(t)): \\
& & \qquad G(t) \\
& & \qquad \&_{\{v\in V\}}\ (v - \sum_{\{a\in A|N(a)=(v,t)\}} E(a)) \geq Max_{\{a\in A_T|N_T(a)=(v,t)\}}\ E_T(a) \\
& & \qquad \&_{\{v\in V\}}\ (v + \sum_{\{a\in A|N(a)=(t,v)\}} E(a)) \leq Min_{\{a\in A_I|N_I(a)=(v,t)\}}\ E_I(a) \\
& & \qquad \rightarrow \\
& & \qquad \&_{\{v\in V\}}\ (v := v - \sum_{\{a\in A|N(a)=(v,t)\}} E(a)) \\
& & \qquad \&_{\{v\in V\}}\ (v := v + \sum_{\{a\in A|N(a)=(t,v)\}} E(a)) \\
& & \qquad |\, t\in T\} \cup \\
& & \{\textit{Meta-rules} \\
& & \;\overline{ev}_1: \\
& & \qquad \overline{G} \\
& & \qquad \&_{\{v\in V\}}\ (v - \overline{exp}_{1,v}) \geq \overline{exp}_{2,v} \\
& & \qquad \&_{\{v\in V\}}\ (v + \overline{exp}_{3,v}) \leq \overline{exp}_{4,v} \\
& & \qquad \rightarrow \\
& & \qquad \&_{\{v\in V\}}\ (v := v - \overline{exp}_{1,v}) \\
& & \qquad \&_{\{v\in V\}}\ (v := v + \overline{exp}_{3,v}); \\
& & \;\overline{ev'}_1: \\
& & \qquad \overline{G'} \\
& & \qquad \&_{\{v\in V\}}\ (v - \overline{exp'}_{1,v}) \geq \overline{exp'}_{2,v} \\
& & \qquad \&_{\{v\in V\}}\ (v + \overline{exp'}_{3,v}) \leq \overline{exp'}_{4,v} \\
& & \qquad \rightarrow \\
& & \qquad \&_{\{v\in V\}}\ (v := v - \overline{exp'}_{1,v}) \\
& & \qquad \&_{\{v\in V\}}\ (v := v + \overline{exp'}_{3,v}) \\
& & \;\rightsquigarrow \\
& & \;//(\overline{ev}_1, \overline{ev'}_1): \\
& & \qquad \overline{G}\ \&\ \overline{G'} \\
& & \qquad \&_{\{v\in V\}}\ (v - \overline{exp}_{1,v} - \overline{exp'}_{1,v}) \geq Max(\overline{exp}_{2,v}, \overline{exp'}_{2,v}) \\
& & \qquad \&_{\{v\in V\}}\ (v + \overline{exp}_{3,v} + \overline{exp'}_{3,v}) \leq Min(\overline{exp}_{4,v}, \overline{exp'}_{4,v}) \\
& & \qquad \rightarrow \\
& & \qquad \&_{\{v\in V\}}\ (v := v - \overline{exp}_{1,v} - \overline{exp'}_{1,v}) \\
& & \qquad \&_{\{v\in V\}}\ (v := v + \overline{exp}_{3,v} + \overline{exp'}_{3,v})\}
\end{array}
$$

Where $Var(t)$ is the set of variables appearing in the guard of transition t, $Type(Var(t))$ is the set of types taken by the set $Var(t)$.

The signature Σ' contains the sort *bool*, the sorts of Σ (the colour set), and the multisets of these sorts, $(\Sigma)_{MS}$. The event sort is ev.

The set of operations is made of operations over the sorts, of events and of operations over events. The sorts are not ordered.

The operations over the sorts are classical operations over the booleans, over the naturals (as the different colours represent different instances of the same natural sort), and over multi-sets. Moreover, three operations over multi-sets are supposed to be available: the $\geq$, Max and the Min operations returning the maximum or minimum value of two given multi-sets. for clarity purpose, all these operations have not been listed as they are classical operations concerning abstract data types.

For each transition $t \in T$ is defined an operation of F, $t : Type(Var(t)) \rightarrow ev$, that takes as parameters the variables $Var(t)$ appearing in the guard of t and returns an event. The operation over the events is the // operation.

There is no ordering defined on the data sort nor on the event sorts.

The set V of local states is given by the set of places of the given CPN. If the place is of colour s, then the corresponding local state is of sort s_{MS}, the multi-set of sort s.

The set X of variables is obtained from the variables in Ax.

The set Ax of axioms contains static axioms, dynamic axioms and meta-rules.

Static axioms, not listed here, concern classical axioms over the sorts.

For each single transition, t, of the CPN is defined a dynamic axiom, whose event part is $t(Var(t))$, the labelled event with its parameters. The precondition of this dynamic axiom is made of three parts: (1) the guard of the transition: $G(t)$; (2) a concatenation of conditions concerning test arcs; (3) a concatenation of conditions concerning inhibitor arcs. We will call these three parts, the *guard part*, the *test* and *inhibitor parts*.

The postcondition is made of two parts: (1) a concatenation of changes of local states corresponding to input arcs; (2) a concatenation of changes of local states corresponding to output arcs. As for the precondition we will call these two parts, the *input*, and *output* part.

Note that we denote by $\&_{\{\ldots\}}$ the concatenation of smf.

The *test part* of the precondition is defined as follows: for each local state $v \in V$, there is a static local axiom verifying if the current value of the local state v minus all the expressions of the input arcs, between v and the transition t, is greater or equal to the maximum expression of the test arcs between v and the transition t. The use of the maximum, Max, operations ensures the tested value to be greater than *all* test arc expressions.

It goes in the same way for the *inhibitor part* of the precondition. The difference with test part is that we ensure that the value of the local state v plus all expressions of the output arcs is less than all expressions of the inhibitor arcs connected to the local state v and the transition t.

These two parts, *test, inhibitor* translate in ACCESS the enabling step conditions 2. and 3. of definition 5.5.

The *input part* of the postcondition is defined as follows: for each local state v, its current value is decreased by the sum of all expressions of input arcs connected to the local state v and the transition t.

It is similar for the *output part* of the postcondition.

These two parts, *input, output* express the changes of marking occurring when t happens, and ensures condition 1. of the enabling step conditions definition 5.5 (a local state cannot be decreased if its value is less than the value of an input arc).

The meta-rule defines the behavior of compound events obtained with the // operator. This meta-rule contains two premices and an action for the compound event.

Each premice is a meta-dynamic axiom having the same structure (three parts in the precondition and two parts in the postcondition) as the dynamic axioms for single transitions explained above. Instead of the sums, and the $Max(\ldots)$ and $Min(\ldots)$ expressions, appearing in the dynamic axioms, we use meta-smf in the premices.

The action is a meta-dynamic axiom having also the same structure (three parts in the precondition and two parts in the postcondition). The *guard part* becomes the concatenation of the two guards of the premices, the *test part* is stronger than those of the premices. We test if the value of each local state v minus the two input expressions $\overline{exp}_{1,v}$, and $\overline{exp'}_{1,v}$ of the premices is greater than the maximum of the two test expressions $\overline{exp}_{2,v}$, and $\overline{exp'}_{2,v}$. It goes in a similar way for the *inhibitor part.* The *input* and *output parts* of the action are given by the combination of the input and output parts of the two premices.

The enabling step conditions of definiton 5.5 imply that all test arcs, adjacent to one of the transitions involved in the step, have their test expression greater than the value of the adjacent place minus all the values of the input expressions of the input arcs adjacent to the place and to a transition involved in the step. It is similar for inhibitor arcs. The input and output arcs adjacent to involved transitions obviously change the markings of the places.

This meta-rule express then these enabling step conditions together with the changes of marking.

The idea is that each dynamic axiom, for single transition of for compound event, is made of the five parts *guard, test, inhibitor, input, output.* When two events occur at the same time (in a compound event), then each part merges the corresponding parts of the two events in order to respect the enabling step conditions and the changes of the markings.

Definition 5.10 *We can define a transformation Tr, that to each CPN with arc extensions, $CPN_{T,I}$ associates the* Access *presentation $PRES'$ of definition 5.9:*

$$Tr(CPN_{T,I}) = PRES'$$

Remark 5.11 *We see easily that Tr is well defined (there is only one* Access *presentation corresponding to a given $CPN_{T,I}$), and is a total function.*

5.4 Discussion

This section briefly discusses the flexibility of the Access approach vs the PN and CPN approaches. Finally, we end with a conjecture asserting that models of CPN and models of their corresponding Access presentation are isomorphic.

Access is more flexible than CPN_{TI}, as the conditions on places are naturally written in the dynamic axioms, while in CPN_{TI} complex use of several different arcs is needed. Moreover, the range of conditions writtable with CPN_{TI} is limited to *lesser or more* comparisons, while it is quite unlimited in Access.

Modifications fo the values of places are realized in PN by adding or decreasing tokens to the current value of the place, it consists in a relative modification. CPN, with arc extension, are able to apply relative and also more absolute modifications to the places as we have seen with the **Clear, Halve, Double** arcs of example 5.7 and this is realized using a combination of arcs. Access, by the mean of a unique SMA, can either define a relative increase or decrease modification to a local state, or clear, halve or double the value of a local state, in addition it can also define any other modification whose operation is defined in the algebraic specification part of the presentation.

The generalization of CPN specifications into Access presentations (definition 5.9) is also maintained at the semantic level. We conjecture that, for a given $CPN_{T,I}$, there is an isomorphism between the models of the $CPN_{T,I}$ and the models of the corresponding Access presentation, $Tr(CPN_{T,I})$.

Conjecture 5.12 *Given $CPN_{T,I}$ a CPN with arc extensions, Tr the transformation of definition 5.10, $PRES'$ the* Access *presentation associated to $CPN_{T,I}$ by Tr, there is an isomorphism, $\cong$, between the set of models corresponding to $CPN_{T,I}$ and the set of models of the corresponding* Access *presentation $PRES'$ when we consider only initial algebra for the algebraic specification part of $PRES'$:*

$$\begin{tikzcd}
CPN_{T,I} \arrow[r, "Tr"] \arrow[d, "\mathrm{Mod}_{\mathrm{CPN}}"'] & PRES' \arrow[d, "\mathrm{Mod}"] \\
\{Models_{CPN}\} \arrow[dr, "\cong"'] & \{Models\} \arrow[d, "\pi"] \\
 & \{Models_{Init}\}
\end{tikzcd}$$

In the diagram above, $\mathrm{Mod}_{\mathrm{CPN}}$, *and* Mod *are the applications associating to $CPN_{T,I}$, $PRES'$, the set of their models $\{Models_{CPN}\}$, $\{Models\}$ respectively; and π is the projection of the set of all models of*

$PRES'$ to the restricted set of models of $PRES'$, whose algebra for the algebraic specification part is the initial one.

Chapter 6

Generalization in Access of specifications written in other languages

Specifications written in some other languages are easily translated into ACCESS presentations. The ideas of the generalizations have been given in [DiMa94] for algebraic specifications, (Algebraic) Petri Nets, Process Algebra [Baet90], Abstract Dynamic Data Types [Aste91], Gamma and CO-OPN. This chapter gives the formal translations for the cases of Algebraic Petri Nets specifications, Gamma programs and CO-OPN specifications. Currently, we give these translations at the syntactic level (the level of the specifications) only. For what is concerning the semantic level, we conjecture the equivalence between the models of a given specification and those of its ACCESS translation.

6.1 Algebraic Petri Nets

Algebraic Petri Nets are Petri Nets whose places take as values an algebraic multi-set, instead of black tokens. This formalism is specially useful as it combines the algebraic specifications, used for describing data structure, and the Petri Nets, used for describing dynamic properties.

Definition 6.1 *[Racl92] An Algebraic Petri Net is a 7-tuple $AN = (P, T, X, FR, \lambda, M_0, Spec)$ with:*

$Spec$	*is an algebraic specification* $Spec = (\Sigma, X', Ax)$ *and* $\Sigma = (S, OP)$ *is the signature*
P	*is a S-set of places*
T	*is a set of transitions, with* $P \cap T = \emptyset$
X	*is a S-set of variables*
FR	*is a flow relation,* $FR \subseteq (P \times T) \cup (T \times P)$
λ	*is an arc inscription respecting the sorts of the places,* λ *is a mapping,* $\lambda : FR \rightarrow T_{m\_\Sigma}(X)$, *and* $\lambda(f) \in (T_{m\_\Sigma}(X))_{m\_\tau(p)}$, $\forall f = (p,t)$ *or* $f = (t,p)$
M_0	*is an initial marking respecting the sorts of the places,* M_0 *is a mapping,* $M_0 : P \rightarrow T_{m\_\Sigma}$, *and* $M_0 \in (T_{m\_\Sigma})_{m\_\tau(p)}$, $\forall p \in P$

Where $T_{m\_\Sigma}(X)$ is the set of terms with variables constructed over the multi-sets of sorts in S, T_{m_Σ} is the same set of terms but with no variables, $\tau(p)$ is the sort $s \in S$ associated to the place p.

Definition 6.2 *Given $AN = (P, T, X, FR, \lambda, M_0, Spec)$ an algebraic petri net, with $Spec = (\Sigma, X', Ax)$, and $\Sigma = (S, OP)$, as in definition 6.1, the corresponding* ACCESS *presentation is given by $PRES =$*

(Σ'', V, X'', Ax'') *where:*

$$
\begin{array}{llll}
1. & \Sigma'' & = & (S'', EV, F) \textit{ with:} \\
 & & & \quad \begin{array}{lll} S'' & = & \{bool\} \cup S \cup (S)_{MS} \\ EV & = & \{ev\} \\ F & = & \{\textit{operations over the sorts } S''\} \cup \\ & & \{t : Type(Var(t)) \rightarrow ev \mid t \in T\} \cup \\ & & \{// : ev\ ev \rightarrow ev\} \end{array} \\
 & & & \textit{with } \subseteq_S = \emptyset,\ \subseteq_{EV} = \emptyset \\
2. & V & = & \cup_{s \in S} V_{s_{MS}} \textit{ where:} \\
 & & & \quad V_{s_{MS}} = \{v \in P \mid \tau(p) = s\} \\
3. & X & = & \{\textit{All variables appearing in } Ax\} \\
4. & Ax'' & = & \{\textit{Static Global Axioms over the sorts } S'' \textit{: } Ax''\} \cup \\
 & & & \{\textit{Dynamic axioms over single transitions} \\
 & & & \ t(Var(t)) : \&_{\{v \in V\}}\,(v := v - \lambda(v,t)) \rightarrow \&_{\{v \in V\}}\,(v := v + \lambda(t,v)) \mid t \in T\} \cup \\
 & & & \{\textit{Meta-rules} \\
 & & & \ \overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1 ; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \quad \leadsto \quad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2\}
\end{array}
$$

Where $Var(t)$ *are the variables appearing in* $\lambda(t)$ *and* $Type(Var(t))$ *are the sorts of these variables.*

The set S'' of data sorts, in the ACCESS presentation, is the extension of the algebraic specification, *Spec*, of the given AN obtained by adding to each sort in S the corresponding multi-set. The local states are nothing else than the places of the AN: if the sort of the place is s in the AN, than the sort of its corresponding local state is m_s, the multiset sort. The static axioms are the equations Ax of the algebraic specification *Spec*. For each transition $t \in T$ of AN, is defined a dynamic axiom, whose precondition is given by a concatenation of *sma*, one for each input arc (v,t), going from a place v to the transition t. These *sma* remove the arc expression $\lambda((v,t))$ from the local state. In a similar way, the postcondition is given by a concatenation of *sma*, one for each output arc (t,v), going from the transition t to a place v, and these *sma* add the arc expression $\lambda((t,v))$ to the local state.

Examples for transformations of Algebraic Petri Nets are close to those for Petri Nets and CPN with arc extensions given in the previous chapter. The only difference consists in the ACCESS signature, Σ, which must contain the same algebraic specification as those of the given algebraic petri net.

6.2 Gamma Language

The Gamma language [Ban93] is a programming style based on multiset transformations. It is useful for describing the logical parallelism occurring in systems and is specially well adapted to reactive systems.

Definition 6.3 *A Gamma program is defined as follows:*

$$
\begin{array}{l}
prog(M) = \Gamma((R_1, A_1), \ldots, (R_m, A_m))(M) \textit{ where for } i = \{1, \ldots, n\}\textit{:} \\
\qquad R_i(x_{i_1}, \ldots, x_{i_n}) \quad \textit{is a first order formulae} \\
\qquad A_i(x_{i_1}, \ldots, x_{i_n}) \quad \textit{is a multi-set}
\end{array}
$$

With M *a multiset and* x_{i_j} *elements of the multiset.*

The pairs (R_i, A_i), $i = \{1, \ldots, m\}$, made of a condition and a multiset, specify the m possible reactions that may occur to the multiset M. The semantics is given by the following definition.

Definition 6.4 *[Ban93] Given* $\Gamma((R_1, A_1), \ldots, (R_m, A_m))(M)$*, a Gamma program as in definition 6.3, its effect on the multiset* M *is the following:*

$$
\begin{array}{lll}
\multicolumn{3}{l}{\Gamma((R_1, A_1), \ldots, (R_m, A_m))(M) =} \\
 & \textbf{if} & \forall i \in \{1, \ldots, m\}, \forall x_{i_1}, \ldots, x_{i_n} \in M, \neg R_i(x_{i_1}, \ldots, x_{i_n}) \\
 & \textbf{then} & M \\
 & \textbf{else} & let\, i \in \{1, \ldots, m\},\ let\, x_{i_1}, \ldots, x_{i_n} \in M,\ \textbf{such that } R_i(x_{i_1}, \ldots, x_{i_n}) \textbf{ in} \\
 & & \Gamma((R_1, A_1), \ldots, (R_m, A_m))(M - \{x_{i_1}, \ldots, x_{i_n}\} + A_i(x_{i_1}, \ldots, x_{i_n}))
\end{array}
$$

If there is a collection of elements $\{x_{i_1}, \ldots, x_{i_n}\}$ of M satisfying a condition R_i, then the multiset M is changed by removing the multiset $\{x_{i_1}, \ldots, x_{i_n}\}$ and by adding the multiset A_i. The program ends when no collection of elements of M fits a condition R_i.

Note that these reactions can occur at any moment and in any order. They may occur more than one time, and the process ends only when no reaction is able to occur.

ACCESS is well suited for the specification of reactive processes, as the semantics given to ACCESS presentations allows any dynamic axiom defined in the presentation to occur at any moment and in any order. At this point, it is easy to associate ACCESS dynamic axioms to Gamma reactions. The following definition of the transformation of Gamma programs into ACCESS presentations describes formally how ACCESS dynamic axioms are used to specify Gamma reactions.

Definition 6.5 *Given $\Gamma((R_1, A_1), \ldots, (R_m, A_m))(M)$ a Gamma program, as in definition 6.3, the corresponding* ACCESS *presentation is given by $PRES = (\Sigma, V, X, Ax)$ where:*

$$
\begin{array}{llll}
1. & \Sigma & = & (S, EV, F) \textit{ with:} \\
 & & & \begin{array}{lll} S & = & \{\textit{All sorts necessary to define the multiset } M \textit{ and the booleans}\} \\ EV & = & \{ev\} \\ F & = & \{\textit{operations over the sorts } S\} \cup \\ & & \{r_i : Type(x_{i_1}, \ldots, x_{i_n}) \rightarrow ev \mid R_i(x_{i_1}, \ldots, x_{i_n}) \in \Gamma, i \in \{1, \ldots, m\}\} \cup \\ & & \{\delta : \rightarrow ev\} \cup \\ & & \{// : ev\ ev \rightarrow ev\} \end{array} \\
 & & & \textit{with } \subseteq_S = \emptyset,\ \subseteq_{EV} = \emptyset \\
2. & V & = & V_{\tau_M} \cup V_{bool} \textit{ where:} \\
 & & & \quad V_{\tau_M} = \{Buffer\}, V_{bool} = \{End\} \\
3. & X & = & \{\textit{All variables appearing in } Ax\} \\
4. & Ax & = & \{\textit{Static Global Axioms over the sorts } S\} \cup \\
 & & & \{\textit{Dynamic axioms over single transitions} \\
 & & & \begin{array}{ll} \{r_i(x_{i_1}, \ldots, x_{i_n}) : & \wedge_{j \in \{i_1, \ldots, i_n\}} (\in_{\tau_M} (x_j, Buffer) = True) \wedge (R_i(x_{i_1}, \ldots, x_{i_n}) = True) \\ & \rightarrow \\ & Buffer := Buffer -_{\tau_M} \{x_{i_1}, \ldots, x_{i_n}\} +_{\tau_M} A_i \mid i \in \{1, \ldots, m\}\} \cup \\ \{\delta : \wedge_{i=\{1,\ldots,m\}} \forall x_{i_1} \ldots \forall x_{i_n}.(& ((\in_{\tau_M} (x_{i_1}, Buffer) = True) \wedge \\ & (\in_{\tau_M} (x_{i_n}, Buffer) = True)) \Rightarrow \\ & R_i(x_{i_1}, \ldots, x_{i_n}) = False) \\ & \rightarrow (End := True)\}\} \cup \end{array} \\
 & & & \{\textit{Meta-rules} \\
 & & & \quad \overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 \quad \rightsquigarrow \quad //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2\}
\end{array}
$$

Where τ_M is the sort of S representing the sort of the multiset M, $\in_{\tau_M}$ is an operation of F specifying if a given element is part of a multiset of sort τ_M.

The set of data sorts contains all sorts and multiset sorts necessary to define the sort of the multiset M, and the sort *bool* for the booleans. The set of event sorts contains only the sort ev. The operations over data and events are operations over data sorts, e.g. operations over multisets and booleans. There are also operations returning events: (1) for each reaction (R_i, A_i) in the Γ program, there is an operation $r_i : Type(x_{i_1}, \ldots, x_{i_n}) \rightarrow ev$ whose parameters, $x_{i_1}, \ldots, x_{i_n}$, are those of the condition R_i (and also those of the action A_i); (2) a special operation δ producing an event with no parameters, which is the event corresponding to the end of the Γ program; (3) the // operation between events, producing compound events.

The set of local states contains two local states: $Buffer$ representing the multiset M, and End representing the state of the program (running or not).

The variables are those used later in the Axioms.

The Static Global Axioms are those needed for defining the operations and the sorts of S, including the $\in_{\tau_M}$ operation.

Dynamic axioms are defined for the labelled events $r_i(x_{i_1}, \ldots, x_{i_n})$ of F and for the δ operation.

The dynamic axioms for $r_i(x_{i_1}, \ldots, x_{i_n})$ have as precondition a *sla* (Static Local Axiom) verifying that each parameter x_{i_k} of the event is in the $Buffer$, it verifies also that with these parameters, the condition R_i evaluates to $True$. The postcondition of these dynamic axioms firstly removes from the local state $Buffer$ the multiset, $\{x_{i_1}, \ldots, x_{i_n}\}$, composed of the parameters, and then adds to $Buffer$ the multiset given by the action A_i. These dynamic axioms specify the reactions (R_i, A_i) as well as their effect on the multiset, i.e. on the local state $Buffer$, according to definition 6.4.

The dynamic axiom for the δ event changes the local state End to $True$ (indicating in that way the end of the Γ program) if all conditions R_i evaluates to false and this for each possible set of parameters x_{i_k} present in $Buffer$.

The meta-rule gives the "true" concurrency for compound events obtained with the // operation.

We will see now, on the basis of an example, how the transformation of definition 6.5 is actually applied.

Example 6.6 *[Ban93] A program testing if a graph is connected or not is given by the following Gamma program:*

$$
\begin{aligned}
&connected(G) = singleton(\Gamma((R_1, A_1), (R_2, A_a))(G))\\
&\quad\quad \textbf{where}\\
&\quad\quad\quad\quad R_1(v, w, (m, n)) = vertices(v) \textbf{ and } vertices(w) \textbf{ and } m \in v \textbf{ and } n \in w\\
&\quad\quad\quad\quad A_1(v, w, (m, n)) = \{v + w\}\\
&\quad\quad\quad\quad R_2(v, (m, n)) = vertices(v) \textbf{ and } m \in v \textbf{ and } n \in v\\
&\quad\quad\quad\quad A_2(v, (m, n)) = \{v\}
\end{aligned}
$$

In this example, a graph G is represented as a multiset of vertices and edges. The pair (m, n) denotes an edge connecting x to y, while v, w denotes sets of vertices. The function $vertices(v)$ is a boolean function testing whether v is a set of vertices or not. Reaction (R_1, A_1) tests if two sets of vertices v, w are connected by an edge (m, n). If this is the case, the edge (m, n) is removed from the multiset G and the two sets v, w are replaced by a larger set $v + w$. The second reaction (R_2, A_2), removes all edges connecting two vertices of a set of vertices.

Both reactions are dedicated to remove from the multiset G all edges connecting set of vertices and to replace all connected set of vertices by only one set. The size of multiset G decreases and G then becomes a singleton if the graph is connected. Finally, the *singleton* function tests if the so obtained multiset G is effectively a singleton (i.e. if G is connected) or not.

The ACCESS presentation for this example follows definition 6.5. The set of local states is $Buffer$, End with $Buffer$ of sort multiset of edges and set of vertices, and End of sort *bool*. The sort of $Buffer$ is denoted by m_s_G, for the multiset constructed over elements of sort s_G, which is our notation for the combined sort of edges and set of vertices. The set of operations F contains, among others, the operations:

$$
\begin{array}{lll}
vertices: & s_G & \rightarrow bool\\
singleton: & m\_s_G & \rightarrow bool
\end{array}
$$

The operation *vertices* takes as parameter an element of the graph (either a set of vertices or an edge) and returns a boolean, while the operation *singleton* takes as parameter a graph and returns a boolean.

The set of dynamic axioms corresponding to the two reactions (R_i, A_i) and to the final event δ are:

$$
\begin{array}{ll}
r_1(v, w, (m, n)): & (\in_{m\_s_G} (v, Buffer) = True) \wedge (\in_{m\_s_G} (w, Buffer) = True) \wedge\\
& (\in_{m\_s_G} ((m, n), Buffer) = True) \wedge\\
& (vertices(v) \wedge vertices(w) \wedge (\in_{vert} (m, v)) \wedge (\in_{vert} (n, w)) = True) \rightarrow\\
& Buffer := Buffer -_{m\_s_G} MAKE_{m\_s_G}(v, w, (m, n)) +_{m\_s_G} MAKE_{m\_s_G}(v +_{vert} w)
\end{array}
$$

$$
\begin{array}{ll}
r_2(v, (m, n)): & (\in_{m\_s_G} (v, Buffer) = True) \wedge (\in_{m\_s_G} ((m, n), Buffer) = True) \wedge\\
& (vertices(v) \wedge (\in_{vert} (m, v)) \wedge (\in_{vert} (n, v)) = True) \rightarrow\\
& Buffer := Buffer -_{m\_s_G} MAKE_{m\_s_G}((m, n))
\end{array}
$$

$$\delta: \forall v_1.\forall w_1.\forall (m_1,n_1).(((\in_{m\_s_G}(v_1, Buffer) = True) \wedge (\in_{m\_s_G}(w_1, Buffer) = True) \wedge$$
$$(\in_{m\_s_G}((m_1,n_1), Buffer) = True)) \Rightarrow$$
$$((vertices(v) \wedge vertices(w) \wedge (\in_{vert}(m,v)) \wedge (\in_{vert}(n,w)) = False)) \wedge$$
$$\forall v_2.\forall (m_2,n_2).(((\in_{m\_s_G}(v_2, Buffer) = True) \wedge (\in_{m\_s_G}((m_2,n_2), Buffer) = True)) \Rightarrow$$
$$((vertices(v) \wedge (\in_{vert}(m,v)) \wedge (\in_{vert}(n,v)) = False))$$
$$\rightarrow (End := True)$$

The $MAKE_{m\_s_G}$ is the operation of F creating a multiset of sort m_s_G from given elements of sort s_G. The $\in_{vert}$ operation tests if a given vertex is element of a given set of vertices.

The dynamic axiom for the reaction (R_1, A_1), (i.e. the dynamic axiom for the labelled event $r_1(v, w, (m, n))$), has a precondition made of a *sla* testing if each parameter $v, w, (m, n)$ is an element of the local state $Buffer$ and if the condition R_1 is satisfied, its postcondition then removes the multiset $\{v, w, (m, n)\}$ from $Buffer$ and adds the multiset given by A_1 (i.e. {v+w}).

The dynamic axiom for the reaction (R_2, A_2), (i.e. the dynamic axiom for the labelled event $r_2(v, w, (m, n))$), is very similar to those for (R_1, A_1), the difference comes from the parameters and the tested condition R_2 and the action A_2.

Finally, the dynamic axiom for the ending event δ has a precondition made of a *sla* testing if firstly R_1 is false for each set of parameters $\{v_1, w_1, (m_1, n_1)\}$ belonging to $Buffer$, and if secondly R_2 is also false for each set of parameters, $\{v_2, (m_2, n_2)\}$ in $Buffer$.

6.3 CO-OPN

CO-OPN is a structured language based on algebraic Petri Nets with adjunction of object-based features and synchronization: object are whole Petri Nets, whose behaviour can be synchronized by the mean of a special operator *with*.

We will not give the formal transformation of CO-OPN specifications into ACCESS presentations, but we will only present informally on the basis of releavant examples, how this transformation happens.

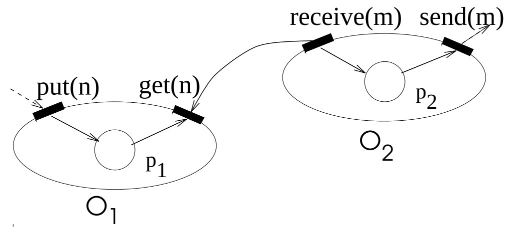

Figure 6.1: Basic CO-OPN example

Figure 6.1 represents two Petri Nets, O_1, O_2, both composed of two transitions and one place. Each of these petri nets is an object and we see, that object O_2 needs to synchronize with object O_1 when transition $receive(m)$ is activated: transition $receive(m)$ can happen only if transition $get(n)$ occurs simultaneously, but $get(n)$ can occur alone even if $receive(m)$ does not happen.

A CO-OPN specification for figure 6.1 is given by:

Object: O_1;
methods: $get : nat, put : nat$;
places: $p_1 : nat$;
var: $n : nat$;
axioms: $t_1 : put(n) : (\ldots) \rightarrow p_1([n])$;
$t_2 : get(n) : p_1([n]) \rightarrow$;
init-marking: $p_1(\emptyset)$;
end;

Object: O_2;
methods: $receive : nat, send : nat$;
places: $p_2 : nat$;
var: $m : nat$;
axioms: $t_3 : receive(m)$ **with** $get(m) :\rightarrow p_2([m])$;
$t_4 : send(m) : p_2([n]) \rightarrow (\ldots)$;
init-marking: $p_2(\emptyset)$;
end;

This specification defines two objects O_1, O_2 (Petri Nets), according to the above figure. It defines for each object, the set of its **methods** (the transitions) and the set of its places. The **axioms** give the flow relation between transitions and places with indication of synchronizations. Axiom t_1 defines the flow relation for the method $put(n)$ with an unknown precondition (before the $\rightarrow$) as figure 6.1 is not complete, and with the postcondition $p_1([n])$ for adding the multiset $[n]$ to the place p_1. Axiom t_2 defines, in a similar way, the flow relation for the method $get(n)$ with the precondition $p_1([n])$ for decreasing the place p_1 from the multiset $[n]$, and with no postcondition. Axiom t_3 concerns method $receive(m)$. It indicates that method $receive(m)$ has to synchronize with method $get(m)$, by the use of the operator **with**. It defines then the behavior of $receive(m)$ with no precondition and with the postcondition $p_2([m])$ for increasing the place p_2 with the multiset $[m]$. The last axio t_4 concerns method $send(m)$, it has the precondition $p_2([n])$ for decreasing p_2 with the multiset $[n]$ and an unknown postcondition.

The corresponding ACCESS presentation will be given by $PRES = (\Sigma, V, X, Ax)$ where:

$$
\begin{array}{llll}
1. & \Sigma & = & (S, EV, F) \text{ with:} \\
 & & & \begin{array}{lll}
 S & = & \{\text{All sorts necessary to define the multiset of } nat\} \\
 EV & = & \{ev\} \\
 F & = & \{\text{operations over the sorts } S\} \cup \\
 & & \{\text{Operations producing events} \\
 & & \begin{array}{ll}
 put: & nat \rightarrow ev \\
 get: & nat \rightarrow ev \\
 receive: & nat \rightarrow ev \\
 send: & nat \rightarrow ev \\
 //: & ev\ ev \rightarrow ev \\
 with: & ev\ ev \rightarrow ev\}
 \end{array}
 \end{array} \\
2. & V & = & V_{m\_nat} = \{p_1, p_2\} \\
3. & X & = & \{\text{All variables appearing in } Ax\} \\
4. & Ax & = & \{\text{Static Global Axioms over the sorts } S \\
 & & & \ (\ldots) \\
 & & & \ \neg(D(with(ev_1, ev_2)))\} \cup \\
 & & & \{\text{Dynamic axioms} \\
 & & & \begin{array}{lll}
 put(n): & (\ldots) & \rightarrow p_1 := p_1 + [n] \\
 get(n): & p_1 := p_1 - [n] & \rightarrow \varepsilon \\
 with(receive(m), get(m)): & \varepsilon & \rightarrow p_2 := p_2 + [m] \\
 send(m): & p_2 := p_2 - [n] & \rightarrow (\ldots)\} \cup
 \end{array} \\
 & & & \{\text{Meta-rules} \\
 & & & \begin{array}{lll}
 \overline{ev}_1 : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & //(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2 \\
 with(\overline{ev}_1, \overline{ev}_2) : \overline{f}_1 \rightarrow \overline{g}_1; \overline{ev}_2 : \overline{f}_2 \rightarrow \overline{g}_2 & \leadsto & \overline{ev}_1 : \overline{f}_1 \& \overline{f}_2 \rightarrow \overline{g}_1 \& \overline{g}_2\}
 \end{array}
\end{array}
$$

The corresponding ACCESS presentation defines all sorts and operations necessary for multisets of *nat* (natural). For each method in the CO-OPN specification is defined an operation producing an event from the parameters of the method. Moreover, two operations producing compound events are defined: // and *with*, for the compound events occuring in parallel and for synchronized events respectively.

The set of local states is nothing else than the set of all places present in the CO-OPN specification: p_1, p_2 whose sorts are the multiset of natural, noted m_nat.

Static Global Axioms are usual axioms necessary for defining correct operations over the multisets of natural, and the additional axiom $\neg(D(with(ev_1, ev_2)))$, indicating that the event $with(ev_1, ev_2)$ is not allowed to happen.

Dynamic axioms are defined for each method, they are constructed in the same way as dynamic axioms for transitions in a basic Petri Net. Note that there is no dynamic axiom defined for $receive(m)$, but there is one for the compound event $with(receive(m), get(m))$. This last dynamic axiom defines this compound event by adding the multiset $[m]$ to the local state p_2.

Meta-rules are defined for the // operator and for the *with* operator. The meta-rule of the // operator defines the parallelism as the "true" parallelism, as we have already seen it some times before. The meta-rule for the *with* operator specifies that behaviour of the first event appearing as parameter of the *with* operator, here it is $\overline{ev}_1$, is defined as the compound behaviour of the $with(\overline{ev}_1, \overline{ev}_2)$ and $\overline{ev}_2$ events. Moreover, the static global axiom $\neg(D(with(ev_1, ev_2)))$ forbids events of the form $with(ev_1, ev_2)$ to happen. The use of this static axiom, together with the meta-rule for the *with* operator, ensures that (1) event $receive(m)$ can happens and event $with(receive(m), get(m))$ cannot happen; (2) the behaviour of $receive(m)$ corresponds to both the behaviour of $get(m)$ and $with(receive(m), get(m))$. This means that event $receive(m)$ leads to an abstraction of event $get(m)$, each time $receive(m)$ happens, the event $get(m)$ happens too.

Remark 6.7 *AS for CPN with arc extensions, we conjecture that there exists an isomorphism between Gamma programs models, CO-OPN models and their models of their corresponding* ACCESS *presentation.*

Chapter 7

Conclusion

The previous chapters were all dedicated to the formal description of ACCESS syntax and semantics, as well as to the possibility for other languages to have their specifications transformed into ACCESS presentations. This section firstly discusses the main features of ACCESS, it then briefly gives some comparison points between ACCESS and some other specification languages, finally it ends with the future possible evolutions of ACCESS.

ACCESS features

ACCESS presentations describe systems by a set of local states, whose value changes under the occurrence of events. Events and Data structure handled by the system are specified as abstract data type. Global constraints on both events and data structure are described with first order formulae (Static Global Axioms), while the behaviour of events is given by causality rules (Dynamic Axioms). A syntactic process (Meta-rules) allows the production, from a restricted set of dynamic axioms, of a large set (possibly infinite) of dynamic axioms.

ACCESS provides a high degree of expressivity combined with a user-defined concurrency instead of a predefined concurrency as it is the case for some other languages. ACCESS offers also some other particularities as a local temporal ordering or non-determinism. These features and some others are discussed below:

Multi-levels of granularity

There is possible access to a given data structure as a whole or to subelements of the data structure. In the following dynamic axiom: $evnt(m) : (first(queue_2) = m) \rightarrow queue_2 := queue_1$ the value of a given element of a queue is tested, and the value of a whole queue is changed with the value of another queue. We have then accessed to an isolated element of a queue, but also to the queue itself.

At the event level we can define all the details of a given event, and we can also define all the details of compound events. But, with meta-rules, we can also define only the global combination of compound events without describing all the internal details.

The use of predicates as static global axioms over events, enables a kind of "modular" definition of events. For example, we can define a "big" event representing a large set of other subevents (combined with a given operator). We define also all the dynamic axioms corresponding to the subevents and to the "big" event. If we add, to the static global axioms, predicates of the form: $\neg(D(ev_i))$ for all subevents, then all the subevents are not allowed to happen. On the contrary, the "big" event can happen. We then have a sort of modular definition for the "big" event as its definition is based on those of the subevents. Another interpretation, we can give to this particularity, is that the "big" event represents an abstraction of all the other subevents: subevents cannot happen alone, but they can happen together in the "big" event; when they happen together none of them is seen separately, only the "big" event is seen.

High expressivity

First order formulae for static axioms lead to high expressivity, as a great variety of properties may be described.

Expressivity is also increased by the possibility to change the value of a local state in a relative but also absolute way. The value of a local state can be decreased (increased) by a given value (as it is the case for algebraic petri nets places), but the value of a local state can also be changed with the value of another local state, for example.

Control of events

The behaviour of a given event is permanently under the control of static local axioms appearing inside the dynamic axioms describing the event. If the conditions are satisfied then the event occurs, otherwise the event fails.

User-defined concurrency

The double view of events, as both (static) data and dynamic entities, allows the definition of any operation on events, whose static properties are described with static axioms and whose behaviour, as event, is given by dynamic axioms and meta-rules. In that manner, it is possible to define different kinds of parallelism (interleaving, "true" parallelism), that can even coexists inside the same system specification: some events may happen together according to a given parallelism, while other events happen together according to another parallelism. Thus, there is no predefined concurrency as it is the case for example with ACP [Baet90] or Petri Nets, which both work with predefined parallelism (interleaving or parallelism given by Petri Nets respectively)

Temporal properties

The pre- and postconditions of dynamic axioms let us define *local* temporal properties, as the postcondition is supposed to happen after the precondition. However, *global* temporal properties are not expressed with ACCESS as no direct mean on the ordering of events is provided. Indirect means could be found, by defining operators of sequentiality between events and by adding appropriate static and dynamic axioms.

Non-determinism

Non-determinism is naturally present in ACCESS, as any of the dynamic axioms can be used at any moment. This similarity between dynamic axioms and reactions in a reactive system, let us suppose that ACCESS is well suited for reactive systems specifications.

The behaviour of an event can be described by more than one dynamic axiom, in that case the behaviour of the event cannot be predictable as it can be the behaviour defined by any of the dynamic axioms.

ACCESS as a generalization of other languages

Our main conjecture 5.12 affirms that there are isomorphisms between models of ACCESS presentations and models of specifications of other languages for which there is a transformation of their specification into an ACCESS presentation. The major implication of this conjecture, if it is true, is that the ACCESS language would be a generalization language, as specifications in other languages are equivalent to specifications given in ACCESS.

Problems

The major problems, currently encountered by ACCESS, are the lack of structure, as no mean is provided for building modular specifications, and the complexity of the definition of ACCESS presentations.

ACCESS vs other languages

The whole family of Petri Nets has a parallelism based on the Petri Nets concurrency: parallelism is given by two or more transitions firable at the same time, a given transition can be fired twice or more at the same time if the value of the adjacent places is sufficient. ACCESS captures the concurrency of Petri Nets and their derived, and goes further as it is more flexible for what concerns the expressivity and the description of events: the variety of conditions writtable in ACCESS is wider than those given by [Jens94]; local states may change their value to any other value.

The Gamma language gives a logical programming style, but we suppose that compilation of Gamma program has to be relatively complex, for example if we consider the large work of matching of parameters. All the elegance of Gamma programs are then paid by this underground. With ACCESS, we are closer to the running program as details have been given about the evolution of the system (e.g. modification of local states), but it is also clear that our specification is heavier than those of Gamma.

All features of the CO-OPN language are easily captured by ACCESS, except the structure (as ACCESS contains no notions of modularity). The reason for this is that CO-OPN is heavily based on algebraic petri nets and that the most extensions of CO-OPN concerning the synchronization between object, realized with the *with* CO-OPN operator of CO-OPN is specified in a natural way with meta-rules in ACCESS.

Future work

Future work will focus on different directions. First of all, the resolution (or not) of conjecture 5.12 is considered, especially the study of isomorphisms between models for the case of Petri Nets, one possible start point could be given by the correspondence of [Muku92] between PN-transition systems and Petri Nets.

Although we have given the syntax and semantics of ACCESS, no interests has yet been given to problems like: initial, terminal algebra; there is, there isn't a model; etc.

As we said above, no structuring of ACCESS presentations is now possible. We intend to add the modularity and hierarchy on the basis of [Guel94] and [Moin91].

Currently dynamic axioms are made of a precondition and of a postcondition. They are of the form: $evnt : f \rightarrow g$. For generalization purposes, and for increasing landmarks in meta-rules, we intend to investigate the possibilities given by meta-rules of the form: $evnt : f \rightarrow g \rightarrow h \rightarrow i \rightarrow \ldots$, with a finite and varying number of parts, each of them separated by $\rightarrow$.

We highlighted previously, the similarity between dynamic axioms and reactions of reactive systems, in order to go further in that direction, we intend to find fields of applications of ACCESS: as for example chemical systems.

Bibliography

[Broy82] M. Broy, M. Wirsing: *Partial Abstract Types*, Acta Infromatica 18, 47-64, Springer-Verlag 1982

[Gogu89] J. A. Goguen, J. Meseguer: *Order-sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions, and Partial Equations*, Computer Science Laboratory, SRI International, Technical Report SRI-CSL-89-10, July, 1989

[Baet90] J.C.M. Baeten and W.P. Weijland : *Process Algebra*, Cambridge Tracts in Computer Science 18, 1990

[Wirs90] M. Wirsing: *Algebraic Specification*, Handbook of theoretical computer science, pp. 675-788, J. Van Leeuwen Editor, Elsevier Science Publishers B. V., 1990

[Bern91] G. Bernot, M. Bidoit: *Introduction aux Spécifications algébriques*, Liens, CNRS URA 1327, Paris, 1991

[Breu91] R. Breu: *Algebraic Specifications techniques in Object Oriented Programming Environments*, Lecture Notes in Computer Science 562, Springer-Verlag, 1991

[Moin91] T. Moineau: *Réutilisation de logiciel: une approche algébrique, son application à Ada et les outils associés*, Université de Paris Sud, Centre d'Orsay, thèse no 1535

[Reis91] W. Reisig: *Petri Nets and algebraic specifications*, Theoretical Computer Science 80 (1-34) 1991.

[Buchs92] D. Buchs, N. Guelfi: *Distributed System Specification Using CO-OPN*, Proceedings of the third workshop on future trends of distributed computing systems, IEEE computer society, 1992

[Muku92] M. Mukund: *Petri Nets and Step Transition system*, International Journal of Foundations of Computer Science Vol.3 No 4 (1992) pp. 443-478

[Racl92] P. Racloz, D. Buchs: *Symbolic Proof of CTL Formulae over Algebraic Nets*, Technical Report, University of Geneva, Cahier du CUI No. 65, 1992

[Ban93] J.P. Banâtre, D. Le Métayer: *Programming by Multiset Transformation*, Communications of the ACM, Vol. 36, no 1, January 1993

[Aste91] E. Astesiano, G. Reggio: *A Structural Approach to the Formal Modelization and Specification of Concurrent Systems* Technical Report 0, Formal Method Group, Dipartimento di Matematica, Università di Genova, Italy, 1991

[Jens92] K. Jensen: *Couloured Petri Nets: Basic concepts, Analysis Methods, and Practical Use*, Vol. 1, Springer-Verlag, 1992

[Jens94] K. Jensen: *A General Systematic Approach to Arc Extensions for Coloured Petri Nets*, Application and Theory of Petri Nets 94, Proceedings, LNCS no 815, Saragoza, 1994

[DiMa94] G. Di Marzo: ACCESS *(AlgebraiC Concurrent Events for System Specification) Un formalisme de spécification intégrant la vraie concurrence*, University of Geneva, Travail de diplôme, 1994

[Guel94] N. Guelfi: Les réseaux algébriques hiérarchiques: un formalisme de spécifications structurées pour le développement de systèmes concurrents, Université de Paris Sud, Centre d'Orsay, thèse no 3313