CIRTA: A FORMAL LANGUAGE FOR MODULAR ECATNETS SPECIFICATION

Résumé

CIRTA ("Construction Incémentale des Réseaux de Petri à Termes Algériques") est un langage de spécification dotant les ECATNets ("Extended Concurrent Algebraic Terms Nets") [6][10] avec des concepts de modularité leur permettant d'être plus appropriés pour des applications réelles. Cet article examine les mécanismes de structuration fournis par CIRTA, pour la conception de systèmes concurrents complexes. Deux techniques de structuration sont présentées. La première se base sur l'utilisation des modules CIRTA qui étendent les ECATNets avec les concepts de nœuds interfaces et de nœuds composés. Le second mécanisme concerne quelques opérations de structuration des modules CIRTA telles que: l’importation, la composition et le renommage. La sémantique de chaque spécification CIRTA utilisant ces mécanismes est définie par la donnée de l’ECATNet ayant le comportement équivalent.

Mots clés: Langage de spécification, réseaux de Petri, ECATNets, spécification algébrique.

Abstract

CIRTA ("Construction Incémentale des Réseaux de Petri à Termes Algériques") is a specification language endowing ECATNets ("Extended Concurrent Algebraic Terms Nets") [6][10] with modularity concepts to make them more suitable for real-world applications. This paper addresses the structuring mechanisms provided by CIRTA, for the design of complex concurrent systems. Two structuring techniques are presented. The first one relies on the usage of CIRTA modules which extend ECATNets with the concepts of interface nodes and composed-nodes. The second mechanism concerns with some structuring operations on CIRTA modules namely: importation, composition and renaming. The semantics of each CIRTA specification using these constructs is defined by giving the behavioral equivalent ECATNet.

Key words: Specification language, Petri nets, ECATNets, algebraic specification.

Concurrent systems are characterized by their dedicated function, real-time behavior, and high requirements on reliability and correctness. In order to devise systems with such features, the design process must be based upon a formal specification that captures the characteristics of concurrent systems. Many computational models have been proposed in the literature to specify such systems, including extensions to finite-state machines, data-flow graphs, and communicating processes. Particularly, Petri nets (PNs) are interesting for the specification of this sort of systems: for instance, they may specify parallel as well as sequential activities and they easily capture non-deterministic behaviors. PNs have been extended in various ways to fit the most relevant aspects of concurrent systems. We can find several PN-based models with different flavors in [2][3][4]. ECATNets [5][8][10] are high-level Petri nets model, in which tokens are algebraic terms [17] holding information. The important intrinsic features of ECATNets are their concurrency and asynchronous nature. These features together with their flexibility have stimulated their application in different areas [5][6][10]. However, the main weakness of classical ECATNets pointed out along the years, is the lack of modularity, forcing the system designer to cope with many details at the same time. In order to develop and analyze complex systems, the system developers need structuring and abstraction concepts that allow them to work with selected part of the specification without being distracted by the low-level details of remaining parts. Therefore, the use of techniques to build up compact specifications through the use of a “divide to conquer” strategy is nowadays, commonly accepted as necessary. Common techniques use different levels of abstraction enabling the construction of the specification in an incremental way. Consequently, for large and complex systems, an adequate specification formalism must deal with modularity concepts. To overcome these limitations of ECATNets, the CIRTA [24] language has recently been introduced as formalism for specifying complex and concurrent systems.

© Université Mentouri Constantine, Algérie, 2005.
It extends ECATNets with module concept and structuring operations so that large systems can be specified and understood stepwisely. The design of complex systems may thus be reduced to the design of simpler and more manageable modules. In this paper, we provide a formal definition of CIRTA modules, and we precise the formal semantics for each structuring mechanism by giving the equivalent (not modular) ECATNet.

- **Paper organization**

  The remainder of this paper is organized as follows. Section 2 gives an overview of ECATNets. Section 3 shows how ECATNets are extended by the concept of CIRTA module. Section 4 defines the syntax and the semantics of some structuring operations on CIRTA modules. In Section 5, an example of specification is presented to illustrate the use of CIRTA language. Section 6 proposes an approach to translate CIRTA specifications into rewriting theories to allow formal analysis of complex concurrent systems. Some concluding remarks are set in Section 7.

### AN OVERVIEW OF ECATNETS

ECATNets are High-Level nets [6][10][8][5] devoted to modeling non-deterministic and concurrent systems. They allow to describe complex systems, using Petri nets [2][19] to model synchronization constraints and abstract data types [17] for specifying the data structures.

#### Definition

An ECATNet is a pair \( \mathbf{E} = (\text{Spec}, \mathbf{N}) \) where \( \text{Spec} = (\Sigma, \text{E}) \) is an algebraic specification and \( \mathbf{N} = (P, T, \sigma, \text{Cap}, \text{IC}, \text{DT}, \text{CT}, \text{TC}, M) \) is a net such that:

- \( \Sigma = (S, \text{Op}) \) where \( S \) is a set of sorts and \( \text{Op} \) is a set of operations.
- \( E \) is a set of \( \Sigma \)-equations.
- \( P \) is a finite set of places.
- \( T \) is a finite set of transitions \( P \cap T = \emptyset \).
- \( \sigma : P \rightarrow S \) is a map which associates a sort to each place.
- \( \text{Cap} : P \rightarrow \{0\} \) is the place capacity function.
- \( \text{IC} : P \times T 
\rightarrow \mathcal{M}(\Sigma) \cup \{\emptyset\} \) is the Input Condition function.
  
  \( \mathcal{M}(\Sigma) \) denotes the set of \( \Sigma \)-terms multisets with variables in \( V \).
- \( \text{DT} : P \times T 
\rightarrow \mathcal{M}(\Sigma) \cup \{\emptyset\} \) specifies the Destroyed Tokens.
- \( \text{CT} : P \times T 
\rightarrow \mathcal{M}(\Sigma) \cup \{\emptyset\} \) is a function defining the Created Tokens.
- \( \text{TC} : T \rightarrow \mathcal{T}_\mathcal{B} \) is the Transition Condition.
  
  \( \mathcal{T}_\mathcal{B} \) is the set of \( \Sigma \)-terms of sort \( \mathcal{B} \) using variables in \( V \).
- \( M : P 
\rightarrow \mathcal{M}(\emptyset) \) is a marking of the net, such that:

\[
\forall p \in P \quad (M(p) \in \mathcal{M}(\emptyset) \land (\sigma(p) = s) \land (|M(p)| \leq \text{Cap}(p))
\]

The graphical representation of a generic net \( \mathbf{N} \) is given in Figure 1, where \( p,s,n \) denotes a place \( p \) of sort \( s \) and capacity \( n \) (we note \( p:s \) a place \( p \) with infinite capacity).

At any time, a transition \( t \) is enabled to fire in making \( M \), when various conditions are simultaneously true. The first condition is that every \( \text{IC}(p,t) \) for each input place \( p \) is enabled (as shown in Figure 2). The second condition is that the transition condition \( \text{TC}(t) \) is true. Finally the addition of \( \text{CT}(p',t) \) to each output place \( p' \) must not result in \( p' \) exceeding its capacity when this capacity is finite. When \( t \) is fired, \( \text{DT}(p,t) \) is removed from the input place \( p \) and simultaneously \( \text{CT}(p',t) \) is added to the output place \( p' \).

#### Example

Figure 3 shows a simple example used to illustrate the main characteristics of ECATNets.

The tokens in the net are of sort \( \text{data} \). The algebraic specification of this sort is given by defining two constants \( (a \text{ and } b) \) and an unary operation \( f \) which semantics is given by two equations \( f(a) = b \) and \( f(b) = a \). The \( \text{Places} \) of the ECATNet, drawn as circles or ellipses, represent the
possible states of the data. The actions performed by this system are indicated by means of rectangles called transitions. Places and transitions are connected by directed arcs, which are annotated by algebraic terms. In the initial state only transition t₁ is enabled (may fire). When t₁ fires, the token (X=a) is removed from place p₁ and simultaneously two tokens b (f(X)=b) are added to p₂. Hence transitions t₁ and t₂ becomes enabled. In commonly known algebraic nets, only one transition in this case is arbitrary selected and fired; while in ECATNets, concurrent as well as sequential firing are possible. Hence after the firing of t₁ in the example of Figure 3, we can have one of the following execution sequences: t₁;t₂ t₂;t₁ t₁//t₂ t₁//t₁ t₂//t₂ (where t₁;t₂ denotes the sequential execution of t₁ and t₂, and t₁//t₂ is the concurrent firing of t₁ and t₂).

CIRTA MODULE

The modularity mechanism is defined in the “common” way used by top-down and bottom-up approaches, supporting refinements and abstractions, and it is based on the concept of module. In CIRTA language, a module is stored in a page. Every page may be used several times in the same specification. The pages with references to a given module M are referred as super-page (upper-level pages) of M, while the pages contained in the module M are referred as sub-pages (lower-level pages).

Intuitively, a CIRTA module contains three types of nodes (a node is either a place or a transition): composed nodes, elementary interface nodes and elementary non-interface nodes. Distinctive graphical notations are used for the representation of each type of nodes as shown in Figure 4.

**Definition**

A CIRTA module is a tuple M=(Spec, Pageset, π, δ) where Spec = (Σ, E) is an algebraic specification and Pageset = (P, T, σ, Cap, IC, DT, CT, TC, M) Such that:

- Σ=(S, Op) where S is a set of sorts and Op is a set of operations.
- P: a finite set of places.
- T: a finite set of transitions P∧T=∅.
- τ: P∧T→{e, c, i} is a map which associates a type for each node of the net:
  - τ(n) = e if n is an elementary non-interface node.
  - τ(n) = c if n is a composed node.
  - τ(n) = i if n is an elementary interface node.
- σ: {p∈P|τ(p)=c}→S is a map which associates a sort to each no-composed place.
- Cap: {p∈P|τ(p)=c}→4-{0} is the place capacity function.
- IC: PxT→mTΣ(V)∪{α′|α∈mTΣ(V)} is the Input Condition.
- DT: PxT→mTΣ(V)∪{∅} is the Destroyed Tokens.
- CT: PxT→mΣ(V) Created Tokens.
- TC: {t∈T|τ(p)=c}→TΣ,boot(V) Transition Condition.
• \( M : \{ p \in P/ \tau(p) \neq c \} \rightarrow mT_\Sigma (\emptyset) \) is a marking of the net, such that: \( \forall p \in \{ p \in P/ \tau(p) \neq c \} \) (\( M(p) \in mT_\Sigma (\emptyset) \)) \& \((\sigma(p)=s)(M(p) \leq \text{Cap}(p))\)

• Pageset: is a finite set of pages (the sub-pages of \( M \))

• \( \pi : \{ n \in P/T/ \tau(n) = c \} \rightarrow \text{Pageset} \) is the map of page assignment.

• \( \delta : \{ n \in P/T / \tau(n) = c \} \rightarrow \text{Br} \) is a port assignment function. It is defined from composed nodes into binary relations such that \( \delta(n) \subseteq 'n' \times \text{Node}(\pi(n)) \) (where 'n' denotes adjacent nodes of n, and Node(\( M \)) is the set of nodes in module \( M \))

**Example**

For instance, let us consider the CIRTA module \( M_1 \) of Figure 5. \( M_1 \) has two elementary interface places \( p_1 \) and \( p_3 \), which are shared with other modules. Apart from \( t_1 \) which is a composed transition, all other nodes are elementary non-interface nodes. The part \( M_2 \) in Figure 6 is a sub-page describing the refinement of the composed transition \( t_1 \), the functions \( \tau, \pi \) and \( \delta \) are defined by:

\[
\begin{align*}
\tau(p_1) &= (t_1) = c, \quad \tau(t_1) = \tau(t_2) = (t_2) = \tau(p_2) = (p_1', (p_2, p_1'))
\end{align*}
\]

**Figure 5:** An example of CIRTA module \( M_1 \).

**Spec \( M_1 \) is**

```
Use M

op f(_: data) : data -> data
vars X,Y : data
eq f(a)=b
eq f(b)=a
end
```

**Spec \( M_2 \) is**

```
sort data

op a :-> data
op b :-> data
op f(_: data) : data -> data
op h(_: data) : data -> data
vars X,Y : data
eq g(X,Y)=h(X)
eq g(a,b)=b
eq g(a,b)=a
eq h(a)=b
eq h(b)=a
end
```

**Figure 6:** A sub-page of the CIRTA module \( M_1 \).

**Dynamic Semantics of a CIRTA module**

The behavior of a CIRTA module is defined here by giving the behavioral equivalent ECATNet. Therefore composed-nodes must be substituted by the associated sub-pages. Each such substitution is a step refinement in the CIRTA specification. The following steps compose the merging process of the sub-page into the super-page:

- References of places and transitions used by the sub-page will be eventually changed in order to produce unique labels;
- One copy of each sub-page is inserted at the super-page; the composed-node is removed;
- The arcs connected with a composed-place are connected to the referred boundary place; for arcs connected with a composed-transition, void boundary places of the sub-page are merged with the associated places at the super-page (arcs and associated arc inscriptions in the sub-page are kept).
- The interface of the sub-page is composed by a set of boundary places or transitions. This set of nodes will constitute the glued points between the sub-page and the super-page, besides the common interface nodes.

Given the CIRTA module \( M_1 \) of Figure.5, we can construct the equivalent ECATNet \( E_1 \) illustrated in Figure7.

**Spec \( E_1 \) is**

```
sort data

op a :-> data
op b :-> data
op f(_: data) : data -> data
op h(_: data) : data -> data
op g(_: data) : data -> data
vars X,Y : data
eq f(a)=b
eq f(b)=a
eq g(X,Y)=h(X)
eq g(a,b)=b
eq g(b,a)=a
eq h(a)=b
eq h(b)=a
end
```

**Figure 7:** Equivalent ECATNet of the CIRTA module \( M_1 \).

**STRUCTURING OPERATIONS ON CIRTA MODULES**

**Importing modules**

A CIRTA module can import another CIRTA module in order to enrich the algebraic specification (with new sorts, operations, and axioms), and/or to extend the net (with new places, transitions, and arcs). The syntax of module importation is
<importation> ::= use <Mod>
Where <Mod> is the identifier of the imported module and use is a CIRTA keyword.
For instance, let \( M_3 \) be the CIRTA module of Figure 5, the module \( M_3 \) in Figure 8 imports \( M_1 \), adds a new operation \( h(\_\_\_) \) and extends the net of \( M_3 \) as follows.

\[
\begin{align*}
\text{Spec } M_3 \text{ is} \\
\text{use } M_1 \\
\text{op } h(\_\_\_) : \text{data } \to \text{data} \\
\text{end}
\end{align*}
\]

\( M_3 \) is equivalent to module \( M_3' \) of Figure 9.

\[
\begin{align*}
\text{Spec } M_3' \text{ is} \\
\text{use } M_2 \\
\text{sort } \text{data} \\
\text{op } f(\_\_\_) : \text{data } \to \text{data} \\
\text{op } h(\_\_\_) : \text{data } \to \text{data} \\
\text{vars } X, Y : \text{data} \\
\text{eq } f(a)=b \\
\text{eq } f(b)=a \\
\text{end}
\end{align*}
\]

\( M_3' \) is an equivalent CIRTA module to \( M_3 \).

Module composition

An important module building operator is composition. It has the syntax

\[
<\text{composition}> ::= \text{make } \langle \text{Mod} \rangle = <\text{Mod}_1> + <\text{Mod}_2> + \ldots + <\text{Mod}_n> \text{ endm}
\]

This structuring operation creates a new module \( <\text{Mod}> \) that combines all the information in its summands \( <\text{Mod}_1>, <\text{Mod}_2>, \ldots, <\text{Mod}_n> \).

Formally, the composition of two modules \( M_1 \) and \( M_2 \) (where for \( i=1 \) to \( 2 \), \( M_i = (\text{Spec}, N_i, \text{Pageset}_i, \pi_i, \delta) \) and \( N_i = (P_i, T_i, \tau_i, \sigma_i, \text{Cap}_i, \text{IC}_i, \text{DT}_i, \text{CT}_i, \text{TC}_i, M_i) \)) is a module \( M \) such that:

- \( M = (\text{Spec}, N, \text{Pageset}, \pi, \delta) \)
  \( N = (P, T, \tau, \sigma, \text{Cap}, \text{IC}, \text{DT}, \text{CT}, M) \)
- \( \Sigma = \Sigma_1 \cup \Sigma_2 \)
- \( E = E_1 \cup E_2 \)
- \( P = P_1 \cup P_2 \)
- \( T = T_1 \cup T_2 \)
- \( \text{Pageset} = \text{Pageset}_1 \cup \text{Pageset}_2 \)
- \( \tau = \tau_1 \cup \tau_2 \)
- \( \sigma = \sigma_1 \cup \sigma_2 \)
- \( \text{Cap} = \text{Cap}_1 \cup \text{Cap}_2 \)
- \( \text{IC} = \text{IC}_1 \cup \text{IC}_2 \)
- \( \text{DT} = \text{DT}_1 \cup \text{DT}_2 \)
- \( \text{CT} = \text{CT}_1 \cup \text{CT}_2 \)
- \( \text{TC} = \text{TC}_1 \cup \text{TC}_2 \)
- \( \delta = \delta_1 \cup \delta_2 \)
- \( \text{TC}(t) = \text{TC}(t)_1 \cup \text{TC}(t)_2 \)
- \( \text{CT}(t) = \text{CT}(t)_1 \cup \text{CT}(t)_2 \)

\section*{Notation}
Let \( f_i : A_i \to B_i \) be functions (\( i=1,\ldots,n \)), we note \( f_1 \circ f_2 \circ \ldots \circ f_n \) the function defined by:

\[
( f_1 \circ f_2 \circ \ldots \circ f_n ) (x) = f_n (f_{n-1} (\ldots (f_1(x)))) \text{ for } x \in A_1 \]

Module renaming

The renaming of a module \( M \) allows to create a new module \( M' \) by changing the notations used in \( M \). The renaming operation uses a set of mappings (also called renaming morphism), namely a sort mapping, an operator mapping, a place mapping and a transition mapping. The syntax of renaming operation is:

\[
<\text{renaming}> ::= \text{Make } <\text{Mod}> = <\text{Mod}>' <\text{renaming morphism}> \text{ endm}
\]

(renaming morphism):=<sort mapping><operator mapping><place mapping><transition mapping>

For example, we rename the CIRTA module \( M_1 \) to get the new module \( M_4 \) as follows.

\[
\text{Make } M_4 = M_1 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \text{ endm}
\]

The module designed by this specification is represented in Figure 10.

\[
\begin{align*}
\text{Spec } M_4 \text{ is} \\
\text{Use } M_2 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \\
\text{vars } X, Y : \text{bool} \\
\text{eq } \text{not}(a)=b \\
\text{eq } \text{not}(b)=a \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Spec } M_4 \text{ is} \\
\text{Use } M_2 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \\
\text{vars } X, Y : \text{bool} \\
\text{eq } \text{not}(a)=b \\
\text{eq } \text{not}(b)=a \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Spec } M_4 \text{ is} \\
\text{Use } M_2 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \\
\text{vars } X, Y : \text{bool} \\
\text{eq } \text{not}(a)=b \\
\text{eq } \text{not}(b)=a \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Spec } M_4 \text{ is} \\
\text{Use } M_2 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \\
\text{vars } X, Y : \text{bool} \\
\text{eq } \text{not}(a)=b \\
\text{eq } \text{not}(b)=a \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{Spec } M_4 \text{ is} \\
\text{Use } M_2 * \langle \text{so } \text{data } \to \text{bool}, \text{op } f(\_\_\_) \to \text{not}(\_\_), \text{tr } t_3 \to \text{inverse} \rangle \\
\text{vars } X, Y : \text{bool} \\
\text{eq } \text{not}(a)=b \\
\text{eq } \text{not}(b)=a \\
\text{end}
\end{align*}
\]
CASE STUDY

In the aim to illustrate the use of structuring mechanism presented in the above sections, we present an example consisting of a simple ring network with four different sites.

- **Module PACK**

  PACK describes the packages which are sent in the network. Each package contains four fields: *n-field* for the package number, *se-field* for the identity of the sender, *re-field* for the identity of the receiver, and *d-field* for the data content of the package. In this module, we define only the data type by an algebraic specification which can be seen as a CIRTA module with an empty net.

  Spec PACK is
  Use DATA NAT SITE-ID
  Sort pack
  op <_,_,_,_>: nat site-id site-id data → pack
  op re(_): pack → site-id
  op se(_): pack → site-id
  op da(_): pack → data
  Vars n: nat s, r: site-id d: data
  eq se(<n,s,r,d>)= s
  eq re(<n,s,r,d>)= r
  eq da(<n,s,r,d>)= d
  end

  The modules DATA NAT and SITE-ID used above are respectively the algebraic specifications of *data, natural numbers* and *site identifiers*. The algebraic specifications DATA and NAT are trivial. They are omitted here since the main principle of the ring network can be understood.

- **Module NETWORK**

  The figure below shows a CIRTA module NETWORK which has four places and four composed transitions positioned in a ring.

- **Module SITE-ID**

  This module contains the elements id₁, id₂, id₃ and id₄ which are used to identify the individual sites of the network.

  Spec SITE-ID is
  Sort site-id
  Ops id₁:: site-id
  id₂:: site-id
  id₃:: site-id
  id₄:: site-id
  end

- **Module SITE**

  In this example we can see that all four composed transitions of NETWORK share the same following CIRTA module SITE.

  The system description will contain a set of eight CIRTA modules.
present site, the package is transferred to the place out and a
copy of the package is put in the place S-ex (indicating that
the package is sent to an external receiver).

Otherwise the package is sent directly to the place R-int
(indicating that the package is received from an internal
sender).

- **Modules SITE₁, SITE₂, SITE₃ and SITE₄**

Each site has the same behavior as SITE. The main
difference concerns the identifier id of the site. Hence each
module SITEᵢ (i=1,...,4) can be obtained by renaming the
module SITE as follows.

The modular specification of the network given above is
behaviorally equivalent to the ECATNet presented in
Figure 11:

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
\text{use PACK} & \\
\text{op id:} & \rightarrow \text{site-id}
\end{align*}
\]

\[
\begin{align*}
\text{Spec SITE is} & \\
Figure 11: The equivalent ECATNet of the module NETWORK.
VERIFICATION OF CIRTA SPECIFICATION

For the levels of complexity typical to concurrent systems, traditional validation techniques like simulation and testing are neither sufficient nor viable to verify their correctness.

Formal methods are becoming an alternative to ensure the correctness of designs. In this section we present a systematic procedure to translate modular CIRTA specifications into rewriting theories.

The advantages of using rewriting logic as a semantic framework for concurrency models has been amply demonstrated in [13][14]. Essentially, rewriting logic has a simple formalism, with only few rules of deduction. It supports user-definable logical connectives, which can be chosen to fit the problem at hand. Besides, it is intrinsically concurrent; and it is realizable in a wide spectrum of logical languages [9][7][15] supporting executable specifications. To verify the correctness of a concurrent system (specified using CIRTA language) we execute the following steps:

• **Step 1**: We translate a CIRTA specification into an ECATNet as stated in definition Section 3.3 and Section 4.

• **Step 2**: We generate a rewrite theory as detailed in [5][11]. Hence, the effect of transitions firing is expressed by rewrite rules which depend strongly on the form of the Input Conditions (IC), and Destroyed Tokens (DT).

• **Step 3**: We use existing analysis tools [9][7][15] to check properties expressed as a rewriting logic formulas.

It should be noted that all the translation steps can be done automatically so that the designer is not concerned with this translation.

CONCLUSION

Our investigation has shown the advantages of using CIRTA; an ECATNets based language, for complex concurrent systems specification. CIRTA allows capturing relevant information characteristics of such systems. In our approach it is feasible to specify large systems as a set of comprehensible structured modules and, at the same time, the essential characteristics of the system may be captured by the model. Moreover we have also presented an example of a practical system specification, namely a ring in order to illustrate the modeling capabilities of CIRTA. To make easy the formal analysis step of concurrent systems specified using CIRTA, we have proposed an approach to translate CIRTA specifications into rewriting theories. Hence, we take advantages of practical tools developed in the rewriting logic framework. In future, we will use CIRTA to develop a formal approach to verification and transformation based synthesis of concurrent systems.

REFERENCES


