Detecting interference through graph reduction

Denis Roegel

▶ **To cite this version:**

Denis Roegel. Detecting interference through graph reduction. [Research Report] Loria & Inria Grand Est. 1997. hal-02435522

HAL Id: hal-02435522
https://inria.hal.science/hal-02435522

Submitted on 10 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Detecting interference through graph reduction

Denis Roegel*
CRIN, BP 239, 54506 Vandœuvre-lès-Nancy cedex, France

Abstract

Parallel programs which run in a shared-memory model have several threads that may interfere. There are constraints between the threads and these constraints can be modelled by a net. We present TLA nets, which are interesting for the representation of concurrent executions. A reduction operation is defined on these nets, in order to detect interferences. These interferences can be eliminated by adding components such as delays to the net. TLA nets are a graphical tool to explore the constraints of parallel programming.

Keywords: graphs, nets, reduction, rewriting, interferences, concurrency, threads, TLA.

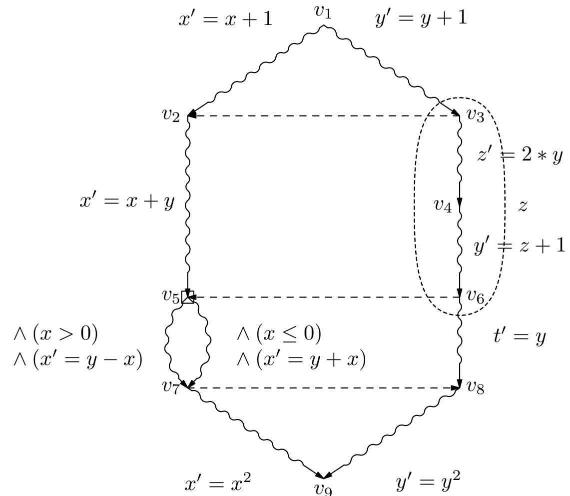


Figure 1: Example of a TNet

1 Introduction

When parallel programs run in a shared-memory model, there are usually several threads and these threads can interfere, thereby producing unwanted side-effects. It is possible to describe the structure of a program segment by a graph whose edges represent the changes in the system, and such graphs will be useful for detecting interferences at early stages. An example of such a graph is given in figure 1.

In this graph, the wavy edges represent instructions, whereas the dashed lines represent dependencies. The wavy edges are labelled by abstract representations of the instructions, namely TLA [3] actions. The parallel branches represent independent actions. All these concepts and notations will be detailed in section 2.1.

First, the formalism of the nets will be precisely presented. Then it will be shown how a network can be reduced and how the reduction makes it possible to detect interferences between different threads.

*Email: roegel@loria.fr.

2 Nets of actions

The nets of TLA actions, the TNets (TLA Nets), are defined here. The presentation is purely syntactic, and the precise semantics is defined by the reduction operation (see section 3). First, an intuitive semantics is given.

TNets are graphs having vertices and edges. There are two kinds of vertices and several kinds of edges. Some edges are labelled by TLA actions, or more precisely, by *contextual TLA actions*. The vertices represent states of a system and the labelled edges are its transitions.

The nets of interest to us will be finite nets: the number of vertices and edges will be finite and the loops will themselves be finite: by that it is meant that there are only a finite number of loops in the construction of the net, and that each of these loops is traversed only a finite number of times during the execution. In section 3, a procedure to eliminate the nets which do not satisfy the constraint of a finite

loop execution (hence, nets which do not *halt*) will be given.

Finally, all the considered TLA actions will be deterministic, that is, they will actually be functional relations.

The structure of the TNets articulates around the following concepts: sequence, concurrency, alternative, delay, synchronization, local variables.

The elements at our disposal to build a TNet make it possible to construct a great many networks. However, only some of these networks are of interest to us.

2.1 Elements of a TNet

The building blocks of a TNet are 1) the edge valued by an *action*, that is, a relation between two consecutive states, and 2) the edge representing a waiting state. A *wavy* edge represents an edge valued by an action, the waving being an image of the *activity*, or more precisely, of the fact that variables can change their values in the corresponding action: $\rightsquigarrow$. For simplicity, an *action* will often be given as a shorthand in place of an *edge valued by an action*. Actions without primed variables, namely predicates, *can* be represented by straight edges: $\longrightarrow$. In general, edges are labelled by an action as it is the case in figure 1.

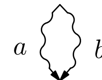
Each vertex represents a set of *local states*. A local state is a valuation of a set of variables. A state can actually be regarded as a control point.

These states are not given *in extenso*, in that the values of the variables are not given explicitely. On the contrary, what defines the valuation of the variables in a state is the way this state is reached. Other formalisms, in particular the very similar notation of Lamport (*predicate/action diagrams* [4]), associate predicates or assertions to the different control points. The presence of assertions corresponds to an operational view, whereas its absence corresponds to a denotational view.

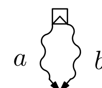
The edges corresponding to actions are directed and link two vertices of a TNet. Intuitively, this means that the action represented by the edge allows the passage from the first state to the second state.

Two vertices can be linked by more than one action. Two cases are distinguished:

- The first case is *parallelism*: a and b occur simultaneously. The superior vertex is of type "AND".



- The second case represents an *alternative*: Here, at most one of the two actions, a or b, can occur. It is possible that none of them is executed, when one considers a fragment of a net made up of two actions among three, for instance. The top vertex is of type "OR" represented by "□". Hence, we always set ourselves in the case of a *deterministic choice*.



Delay and synchronization are indicated using simple and double dashed arrows. They introduce an ordering constraint between two actions.

Some variables can be local to several actions. This fact is represented by surrounding the set of concerned actions by a dotted line labelled with the local variable (see figure 1). We add as a constraint that the intersection of two dotted areas is necessarily one of these areas, which forbids nets where actions bound by two different local variables overlap.

The graphs under consideration will always have a source and a sink. This is natural since these concepts correspond to the initial and final states of the execution of a function.

However, many nets obtained by a combination of the primitive elements will give rise to problems we will try to *detect* and *solve* in section 3.

2.2 Formalization

The concepts that have been described in the previous section are formalized here.

Definition 1 (deterministic action) *An action $A(x, x')$ is said to be deterministic if the relation represented by A is functional, that is, if x' is a function of x.*

For instance, $x' = x + 1$ (x' is the value of x in the next state) is deterministic, but $x' + y' = x + y$ is not.

Definition 2 (set of actions) *We assume* Act *to be a set of deterministic TLA actions. This set will not be defined formally.*

Definition 3 (variables of an action) *The set of variables of an action is the union of the set of* used *variables (unprimed) and the set of* modified *variables (primed):* $Var(a) \triangleq \mathfrak{U}(a) \cup \mathfrak{M}(a)$.

x' *is always included in* $\mathfrak{M}(a)$ *even if* x *is not modified. One can view* $\mathfrak{M}(a)$ *as the set of variables* constrained *by* a.

Definition 4 (set of variables) *Let* $\mathfrak{V}$ *be a set of variables such that* $\forall a \in \mathsf{Act}, Var(a) \subseteq \mathfrak{V}$.

Definition 5 (set of values) *We assume a set of values* Val.

We will see in section 3 that the *environment* of the net must be taken into account for the *reduction* of the nets. The interaction between a net and its environment is based on the interaction between an action and the environment. Henceforth, we will add to each action some *requirements* with respect to the environment. In return, this will allow us to deduce properties that the net will guarantee. This will be detailed later.

Definition 6 (contextual actions) *Let* $\mathbf{CA} = (\mathsf{Act} \times \mathcal{P}(\mathfrak{V})) \cup \{\langle \bot, \bot \rangle\}$; $\mathbf{CA}$ *represents the set of actions with a condition on the context;* $\langle a, r \rangle \in \mathbf{CA}$ *if* r *is a set of variables which must not be modified by the environment. In general,* $\langle a, r \rangle = \langle a, \mathfrak{U}(a) \rangle$ *where* $\mathfrak{U}(a)$ *is the set of variables used in* a.

Definition 7 (set of dependencies) $D = \{\bot, \dashrightarrow, \dashleftarrow, \leftrightarrow\}$ *is the set of dependencies. The elements of* D *must be distinct from the elements of* Act. $\bot$ *will represent the absence of any link between two vertices of a net.* $\dashrightarrow$ *and* $\dashleftarrow$ *will represent the delay and* $\leftrightarrow$ *the synchronization.*

The nets can now be defined:

Definition 8 (net) *A* GTNet *is a tuple* $\langle V, br, edge, loc \rangle$ *where:*

- V *is a finite set of vertices;*
- $br : V \to \{\cdot, \Box\}$ *is the branching function:* $br(s) =$ "$\cdot$" *if* s *is an "AND" vertex and* $br(s) =$ "$\Box$" *if it is an "OR" vertex.*
- $edge : V^2 \to (\mathcal{P}(\mathbf{CA}) \cup D)$ *is the "edge" function.* $edge(v_1, v_2) \in \mathcal{P}(\mathbf{CA})$ *if* v_1 *and* v_2 *are connected by TLA actions.* $edge(v_1, v_2) \in D$ *if* v_1 *and* v_2 *are simply linked by dependencies, or are not linked at all.*
- $loc : \mathfrak{V} \to \mathcal{P}(V \times V \times \mathbf{CA})$ *is the function which associates to each variable a set of edges where this variable is local. The set of edges is* $\{(v_1, v_2, ar) \in V \times V \times \mathbf{CA} / ar \in edge(v_1, v_2)\}$.

  If $loc(v_1)$ *and* $loc(v_2)$ *are the edges where* v_1 *and* v_2 *are local variables, then either* $loc(v_1) \cap loc(v_2) = \varnothing$*, or* $loc(v_1) \subseteq loc(v_2)$*, or* $loc(v_1) \supseteq loc(v_2)$*. This constraint may make it necessary to rename some variables.*

Definition 9 (source of a net) v *is a source if* $v \in V$ *and* $\forall u \in V : edge(u, v) \notin \mathcal{P}(\mathbf{CA})$.

Definition 10 (sink of a net) v *is a sink if* $v \in V$ *and* $\forall u \in V : edge(v, u) \notin \mathcal{P}(\mathbf{CA})$.

Definition 11 *A* TNet *is a* GTNet *with a unique source and a unique sink. The sink* v *is such that* $br(v) = \cdot$.

Definition 12 (action escaping from a vertex) *The action* a *is said to* escape *a vertex* s *if*

$$\exists r \in \mathcal{P}(\mathfrak{V}) : \langle a, r \rangle \in \bigcup_{u \in V} edge(s, u)$$

2.3 Example of a formal representation of a net

Consider figure 1. This net has nine vertices denoted by $v_1, v_2, \ldots, v_9$. We have:

- $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$.
- $\forall i \neq 5 : br(v_i) = \cdot$ and $br(v_5) = \Box$.
- edges: $edge(v_1, v_2) = \{\langle x' = x + 1, \{x\} \rangle\}$, $edge(v_2, v_3) = \dashleftarrow$, $edge(v_5, v_7) = \{\langle (x > 0) \wedge (x' = y - x), \{x, y\} \rangle, \langle (x \leq 0) \wedge (x' = y + x), \{x, y\} \rangle\}$, etc.
- $loc(z) = \{(v_3, v_4, \langle z' = 2 * y, \{y\} \rangle), (v_4, v_6, \langle y' = z + 1, \{z\} \rangle)\}$
- v_1 is the source and v_9 is the sink.

3 Reduction of a net

A TNet reduction will replace a complex net by a net composed of a unique edge which is equivalent to it (in some sense). In other words, the reduction associates a TLA action to a net. This operation modifies the granularity of a net.

Moreover, the reduction makes it possible to separate the nets into two categories: those that can be reduced into an action and those that cannot. The case of nonreducible nets will be studied in more detail.

Underspecified nets have also to be completed. For instance

$$x' = x + 1$$
$$y' = x + z$$

poses problems because it is not known how the value of z changes. The solution is twofold:

- *All unprimed variables*[1] *in the second action must appear in the first action, in their primed form, possibly meaning the identity.*
- *If an action uses a variable x, it must also mention x'.*

If these conditions are not respected, the reduction of these two actions is unknown by convention, and is denoted by $\bot$.

These rules will be formalized in the next section.

3.1 Formalization

The previous section explained intuitively how the net reduction is worked out, and the need to introduce additional constraints if we want the net to be reduced in a specific way.

From now on, TLA actions will no longer be manipulated. Instead, contextual actions (cf. §2.2) are used in order to handle these constraints.

The operations $\wedge_c$, $\vee_c$ and $\bullet_c$ are defined on the elements of **CA**, the set of contextual actions. They are the contextual equivalents of $\wedge$, $\vee$ and $\bullet$.

Definition 13 (contextual conjunction) *$\langle a_1, r_1\rangle \wedge_c \langle a_2, r_2\rangle$ is defined as follows:*

- *if $a_1 = \bot$ or $a_2 = \bot$, then $\langle a_1, r_1\rangle \wedge_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$ (failure propagation)*
- *if $\mathfrak{M}(a_1) \cap r_2 = \varnothing$ and $\mathfrak{M}(a_2) \cap r_1 = \varnothing$, then $\langle a_1, r_1\rangle \wedge_c \langle a_2, r_2\rangle \triangleq \langle a_1 \wedge a_2, r_1 \cup r_2\rangle$*
- *otherwise, $\langle a_1, r_1\rangle \wedge_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$*

[1] These variables can also occur as primed variables in the second action, of course.

As soon as a reduction yields $\langle \bot, \bot\rangle$, the reduction of the whole TNet is undefined. The result of the execution is ignored: the TNet must be better specified.

Definition 14 (contextual disjunction) *$\langle a_1, r_1\rangle \vee_c \langle a_2, r_2\rangle$ is defined as follows:*

- *if $a_1 = \bot$ or $a_2 = \bot$, then $\langle a_1, r_1\rangle \vee_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$ (failure propagation)*
- *if $a_1 \Rightarrow \neg a_2$ then*

  $\langle a_1, r_1\rangle \vee_c \langle a_2, r_2\rangle \triangleq \langle a_1 \vee a_2, r_1 \cup r_2\rangle$

  (the dependencies are determined statically)
- *otherwise, $\langle a_1, r_1\rangle \vee_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$*

  (exclusion of non-determinism)

Definition 15 (contextual sequentialization) *Finally, $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle$ is defined as follows:*

- *if $a_1 = \bot$ or $a_2 = \bot$, then $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$ (failure propagation)*
- *if $\neg(\mathfrak{U}(a_1) \subseteq \mathfrak{M}(a_1))$ then $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$*
- *if $\neg(\mathfrak{U}(a_2) \subseteq \mathfrak{M}(a_2))$ then $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$*
- *if $\mathfrak{M}(a_1) \neq \mathfrak{U}(a_2)$ then $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle \triangleq \langle \bot, \bot\rangle$*
- *otherwise $\langle a_1, r_1\rangle \bullet_c \langle a_2, r_2\rangle \triangleq \langle a_1 \bullet a_2, r_1 \cup r_2\rangle$*

Theorem 1 (Associativity of $\wedge_c$, $\vee_c$ and $\bullet_c$) *The three operations $\wedge_c$, $\vee_c$ and $\bullet_c$ are associative. The proofs are given in [6].*

Theorem 2 (Commutativity of $\wedge_c$ and $\vee_c$) *Obvious.*

3.2 Reduction rules

Reduction rules in which the actions are accompanied by an hypothesis on the environment (*requirement*) are stated here. Each action is therefore a pair $\langle$*guarantee*, *rely*$\rangle$, made of an hypothesis that the environment must satisfy (*rely*) and of an action (*guarantee*) which is satisfied if the first hypothesis is true. These pairs will be manipulated in a compositional manner, in that the pair of a complex net will be obtained from the pairs corresponding to its parts. Reference [1] gives other approaches of this paradigm for the construction of larger scale specifications.

The rules enable us to handle the interferences occuring when actions are composed.

The rules are given in table 1.

The rules involve the contextual composition operations $\bullet_c$, $\wedge_c$ and $\vee_c$. Each rule has a non-contextual counterpart, obtained by replacing $\bullet_c$ by $\bullet$, $\wedge_c$ by $\wedge$ and $\vee_c$ by $\vee$.

The rules R_1 to R_4 have a very simple expression, in spite of the fact that they take into account the possible interferences. These interferences are handled by the contextual connectives on the contextual actions. Some of the rules are explicited below.

- Rule R_1 :

  if $\exists s_1, s_2 \in V, ar_1, ar_2 \in \mathit{edge}(s_1, s_2)$ such that $br(s_1) = \cdot$, then

  - for all s_a, s_b: $\mathit{edge}'(s_a, s_b) \triangleq \mathit{edge}(s_a, s_b)$ if $s_a \neq s_1$ or $s_b \neq s_2$;
  - $\mathit{edge}'(s_1, s_2) \triangleq (\mathit{edge}(s_1, s_2) \setminus \{ar_1, ar_2\}) \cup \{ar_1 \wedge_c ar_2\}$

- Rule R_5 :

  Condition : no contextual action other than a is supposed to escape from the "AND" node.

- Rule R_6 : "repeat until"

  if $\exists s_1, s_2, s_3 \in V$ such that

  1. $\mathit{edge}(s_1, s_2) = \{\langle a, r_a \rangle\}$,
  2. $\mathit{edge}(s_2, s_3) = \{\langle b, r_b \rangle\}$,
  3. $\mathit{edge}(s_2, s_1) = \{\langle c, r_c \rangle\}$
  4. $br(s_2) = \square$
  5. $\forall s : (\mathit{edge}(s_1, s) \in \mathcal{P}(\mathbf{CA})) \Rightarrow ((s = s_2) \wedge \mathit{edge}(s_1, s) = \{\langle a, r_a \rangle\})$
     (No action other than a can escape from a top node)
  6. $\mathit{Pred}(c)$ (c is a predicate)
  7. $\mathfrak{U}(c) \subseteq \mathfrak{M}(a)$ (the variables of c must be primed in a)
  8. $c \Rightarrow \neg b$
  9. $b \vee c$

  and if (with [7]'s notations for the denotational semantics)

  1. $\mathit{repeat}\ \sigma \triangleq (\mathcal{E}[\![\neg c]\!]\sigma \rightarrow \sigma, \mathit{repeat}(\mathcal{C}[\![a]\!]\sigma))$
  2. $ru(a, c)(\sigma) \triangleq \mathcal{C}[\![\mathbf{repeat}\ a\ \mathbf{until}\ \neg c]\!]\sigma = \mathit{repeat}(\mathcal{C}[\![a]\!]\sigma)$ where $ru(a, c)(\sigma) = \perp$ if the loop diverges;

  then:

  - $\mathit{edge}'(s_2, s_1) = \perp$
  - $\mathit{edge}'(s_1, s_2) = \mathcal{D}(\langle a, r \rangle, c)$ where
    - if $\forall \sigma, ru(a, c) \neq \perp$, then $\mathcal{D}(\langle a, r \rangle, c) \triangleq \langle ac, r \rangle$ with $ac \triangleq \sigma' = ru(a, c)(\sigma)$
    - if $\exists \sigma, ru(a, c) = \perp$, (case of divergence) then $\mathcal{D}(\langle a, r \rangle, c) \triangleq \langle \perp, \perp \rangle$

  $\mathcal{D}(a, c)$ is the relation associated to the function $ru(a, c)$.

  The semantics of a loop is an action if the number of loops is finite.

- Rule R_8 :

  Conditions:

  1. the contextual actions a_2 and b_2 escaping from the intermediate vertices are the only ones escaping from these vertices.
  2. a_1 and b_2 do not interfere.

3.3 Confluence

It will be shown that the rewriting system made of rules R_1 to R_9 is confluent. First, a few notions useful for the proof are introduced.

3.3.1 Complexity of a net

The complexity $\mathit{comp}(R)$ of a net R is defined by

$$\mathit{comp}(R) = n_a + n_\vee + n_\leftrightarrow + 2 \cdot n_\dashrightarrow + n_l$$

where n_a is the number of edges representing actions, $n_\vee$ is the number of "OR" vertices, $n_\leftrightarrow$ is the number of double dependencies, $n_\dashrightarrow$ is the number of simple dependencies and n_l is the number of localities. The weights are different for $n_\leftrightarrow$ and $n_\dashrightarrow$ because rule R_8 does not diminish the number of edges.

Notice that there exists a net of complexity n, for any $n > 0$. It is also obvious that each reduction rule decreases the complexity of a net. For instance, for $(1 \leq i \leq 9) \wedge (i \neq 6)$, $\mathit{comp}(R_i(R)) = \mathit{comp}(R) - 1$. And $\mathit{comp}(R_6(R)) = \mathit{comp}(R) - 2$.

Since the nets under consideration are finite, the reduction process ends. It now remains to show that the reduction is confluent.

$R_1 :\ a\ \ b \xrightarrow{R} a \wedge_c b$	$R_2 :\ a\ \ b \xrightarrow{R} a \bullet_c b$
$R_3 :\ a\ \ b \xrightarrow{R} a \vee_c b$	$R_4 :\ a\ \ b \xrightarrow{R} a \bullet_c b$
$R_5 :\ a \xrightarrow{R} a$	$R_6 :\ \langle a, r_a \rangle\ \ \langle c, r_c \rangle\ \ \langle b, r_b \rangle \xrightarrow{R} \langle \mathcal{D}(a,c), r_a \rangle\ \ \langle b, r_b \rangle$
$R_7 :\ \langle a, r \rangle\ \ v \xrightarrow{R} \langle \exists v : a, r \backslash \{v\} \rangle$	$R_8 :\ a_1\ b_1\ a_2\ b_2 \xrightarrow{R} a_1\ b_1\ a_2\ b_2$
$R_9 :\ a_1\ b_1\ a_2\ b_2 \xrightarrow{R} a_1\ b_1\ a_2\ b_2$	

Table 1: Reduction rules

3.3.2 Confluence

Let $\rightarrow^*$ be the reflexive transitive closure of some binary relation $\rightarrow$.

Definition 16 (Normalization) *A net G is normalizable if there exists a reduction $G \rightarrow^* G'$ where G' is not reducible.*

Definition 17 (Confluence) *The rewriting relation $\rightarrow$ is confluent in x, if for all x_1, x_2, if $x \rightarrow^* x_1$ and $x \rightarrow^* x_2$, there exists x' such that $x_1 \rightarrow^* x'$ and $x_2 \rightarrow^* x'$.*

Theorem 3 (Unicity of normal form) *A net G has a unique normal form.*

The proof is based on the locality of the rewriting rules and the permutability of the rules in some cases. The proof is by induction on the complexity of the net. See [6].

The unicity of the normal form is true even for nets which do not reduce to actions.

Theorem 4 (Confluence) *The rewriting relation $\xrightarrow{R}$ is confluent.*

This is an immediate consequence of the unicity of the normal form.

Since the $\xrightarrow{R}$ relation is confluent, a new rewriting relation $\xrightarrow{R*}$ can be defined. It represents the iterated relation of the $\xrightarrow{R}$ relation.

Definition 18 (Confluence rewriting) *$N_1 \xrightarrow{R*} N_2$ if $\exists M_1, \ldots, M_n$ such that $N_1 = M_1$, $N_2 = M_n$, $M_i \xrightarrow{R} M_{i+1}$ and $\neg \exists M_p$ such that $M_n \xrightarrow{R} M_p$.*

3.4 Observational semantics of a TNet

The observational semantics of a TNet is defined in the following way:

Definition 19 (Observational semantics) $[\![.]\!] : \mathbf{Net} \rightarrow \mathbf{CA}$
where

- $[\![N]\!] \triangleq \xrightarrow{R*}(N)$ *if* $\xrightarrow{R*}(N) \in \mathbf{CA}$
- $[\![N]\!] \triangleq \langle \bot, \bot \rangle$ *otherwise; (this case occurs when none of the rules can be applied)*

and

- **CA** *is a set of contextual actions (see definition 6).*
- **Net** *is the set of nets.*

Because of its *global* aspect, the observational semantics is of course similar to the denotational semantics.

3.5 Non-reducible nets

The rules seen in section 3.2 make it possible to reduce a net when the rules are read from left to right. However, if they are read from right to left, they represent *construction* rules.

The RTNets are defined as TNets whose reduction is a TLA action.

Definition 20 (generated nets) $\mathsf{RTNets} = \{t \in \mathsf{TNets} / \xrightarrow{R*}(t) \in \mathsf{Act}\}$

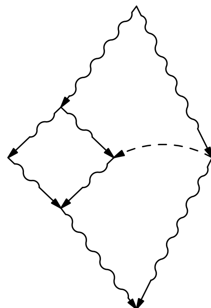


Figure 2: Non reducible net

It is easy to find TNets which are not reducible, and thus, which are not RTNets. Almost all TNets are not reducible, which is easily understood, since a program written randomly is very unlikely to do something meaningful (or, let us say, interesting).

For instance, the net of figure 2 cannot be reduced to an action and can therefore not be generated by the inverse rules.

4 Related work

A net formalism has often been employed in order to represent and analyze concurrent systems. Rabinovich [5] presents ideas which have much in common with ours. He introduces a notion of nets made from two kinds of vertices: circles which represent *places* and squares which represent *ports*. The edges are called *channels*. Ports correspond to data or variables and places to operations on these data.

Rabinovich considers such nets where a function is associated to each place output. These functions take the data from input ports and compute the output. If the ports and internal places of the net are hidden, a solution to the equations determined by the functions is a function determining the outputs in terms of the inputs. This is what Rabinovich calls the *observable relation*. The places can also be labelled by *relations*.

The TNets are different from Rabinovich nets in that all vertices in TNets represent states, corresponding therefore to ports in Rabinovich's formalism. The edges of a TNet correspond to the functions labelling Rabinovich places. Furthermore, the TNets have two kinds of branching at the vertices and have notions of dependencies and local variables.

The "reduction" of a Rabinovich net is a solution to the functional or relational equations defined by the places. These equations are not conditional, as in the TNets, since there are no dependencies or choices. Moreover, there is no possibility in Rabinovich's nets to have two branches in parallel, that is to have a port which would be obtained from two functions. So, there is no notion of interference, as is the case with the TNets, but there might be no solutions to the equations defined by the net. Rabinovich doesn't give a rewriting system, but could define the reduction of a system in that way.

In [4], Lamport defines the *predicate/action diagrams* which are graphs labelled by a property (predicate) [2] and represent the states satisfying this property, and whose edges are actions. A predicate/action diagram denotes a TLA formula. The purpose of this formalism is to give a graphical view of some components of a system, for instance by restricting ourselves to the behaviour of some variables.

The formalism is hence very close to the TNets' formalism, but Lamport does not study the reduction of these nets. In the TNets, the vertices are not labelled by predicates because we are not interested by a local study of the net, but by a global study.

5 Conclusion

The reduction of TLA nets proves to be interesting for the detection of interferences in parallel executions. The reduction is based on a few rewriting rules and is confluent. Hence, a well-defined net has a clear semantics which is its reduction. The use of a graphical representation helps linking the net with a real execution.

References

[1] Pierre Collette and Cliff B. Jones. Enhancing the Tractability of Rely/Guarantee Specifications in the Development of Interfering Operations. Technical Report UMCS-95-10-3, Department of Computer Science, University of Manchester, 1995.

[2] Leslie Lamport. Control Predicates are Better than Dummy Variables for Reasoning about Program Control. Technical Report 11, DEC-SRC, Palo Alto, California, USA, 1986.

[3] Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[4] Leslie Lamport. TLA in Pictures. In Guy E. Blelloch, K. Mani Chandy, and Suresh Jagannathan, editors, *Specification of Parallel Programs: Proceedings of the DIMACS Workshop*, number 18 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 293–307. American Mathematical Society, 1994.

[5] Alexander Rabinovich. Modularity and Expressibility for Nets of Relations, 1996.

[6] Denis Roegel. *Étude de la sémantique de programmes parallèles « réels » en TLA*. Thèse d'université, Université Henri Poincaré — Nancy 1, 1996. Defended on November 7, 1996. Also CRIN reports 96-T-214 (french) and 97-R-125 (english). Available at `http://www.loria.fr/~roegel`.

[7] R. D. Tennent. The Denotational Semantics of Programming Languages. *Communications of the ACM*, 19(8):437–453, August 1976.