Well-structured Petri Nets extensions with data

Remi Bonnet
LSV – ENS Cachan

Master Thesis – 12 March 2010

Contents

Chapter 1

Introduction

Well-structured transitions systems (WSTS) are a general class of infinite state systems for which decidability results rely on the existence of a well-quasi-ordering that is compatible with the transitions. Many models can be seen as WSTS : lossy counter machines [1], lossy channel systems, [12], string rewrite systems [8] are for example WSTS.

Petri Nets are another famous model that can be seen as a WSTS. Since the introduction of Petri nets [19], many extensions have been proposed, most of them being well-structured. One of the recent ones are data nets [17], that subsumes almost all already proposed extensions. The focus of our work was to try to find the limits of data nets, and to try to distinguish them from other extensions. As a secondary goal, we wanted to provide an alternate representation of data nets to ease the comprehension of this model.

Chapter 2

Overview

Section 3 recapitulates many results about well-orders and Petri Nets. We introduce there a few classical extensions of Petri Nets that are well-studied in the litterature and we define the notions of similarity that we will use to relate our different extensions in the following sections.

Section 4 introduces a new model, communicating affine nets. We define it in a way that it is easy to add or remove power from it, the aim being to have a flexible, modular model, that can be made equivalent to data net for some precise set of variations.

Section 5 looks at the data net model itself. We investigate how the number of transitions and the arity are related, then we show how data nets are equivalent to some flavor of communicating affine nets.

Section 6 and section 7 try to find new results on how the different extensions are separated. We will look here at both reasoning about the coverability sets and differences between the recognized languages of the different extensions.

Chapter 3

Preliminaries

3.1 Well-orders

Orders A *quasi-ordering* (a *qo*) is a reflexive and transitive relation $\leq$ on a set X. We use $x < y$ to denote that $x \leq y \not\leq x$. A *partial ordering* (a *po*) is an antisymmetric qo. Any qo induces an equivalence relation ($x \equiv y$) and a partial ordering between the equivalent classes.

Here is a few results from the theory of well-ordering (see also [16], [13]) :

Definition 3.1.1. A *well-quasi-ordering* (a *wqo*) is any quasi-ordering $\leq$ (on a set X) such that, for any infinite sequence $x_0, x_1, x_2...$, there exists $i < j$ with $x_i < x_j$.

Given $\leq$ a quasi-ordering, an *upward-closed set* is any set $A \subset X$ such that $y \geq x \in A$ entails $y \in A$. To any $A \subset X$, we associate $\uparrow A = \{y \mid \exists x \in A.\ y \geq x\}$, the *upward closure* of A. A *basis* of an upward closed set A is a set B such that $A = \uparrow B$. *Downward-closed sets* and the *downward closure* $\downarrow A$ are the dual notions.

Lemma 3.1.2. *[13] If $\leq$ is a wqo, then any upward closed set I has a finite basis.*

Pointwise comparison $\leq$ in $\mathbb{N}^m$ is a well-order.

Multisets, Words Given A a set, we denote by $A^{\oplus}$ the multisets of A, and A^* the words of A. Multisets will be written in the same way as words, as a sequence of elements $x_1...x_n$, with $x_i^{n_i}$, $n_i \in \mathbb{N}$ being a shortcut for $x_i.x_i..n$ times$..x_i$

The *subword embedding comparison* $\preceq_w$ on A^* with an underlying order $\leq_A$ on the letters is defined by :

$$u_1...u_m \preceq v_1...v_n \Leftrightarrow \exists \varphi : 1..m \rightarrow 1..n \left\{ \begin{array}{l} \varphi \text{ strictly increasing} \\ \forall 1 \leq i \leq m.\ u_i \leq_A v_{\varphi(i)} \end{array} \right.$$

The *multiset embedding comparison* $\preceq_s$ on $A^{\oplus}$ is similarly defined by :

$$u_1...u_m \preceq v_1...v_n \Leftrightarrow \exists \varphi : 1..m \rightarrow 1..n \left\{ \begin{array}{l} \varphi \text{ injective} \\ \forall 1 \leq i \leq m.\ u_i \leq_A v_{\varphi(i)} \end{array} \right.$$

Assuming $(A, \leq_A)$ is a well-ordered set, A^* and $A^{\oplus}$ are respectively well-ordered by $\preceq_w$ and $\preceq_s$

3.2 Labeled Transition Systems

A *labeled transition system* (LTS) $\mathcal{S} = \langle S, s_0, \Sigma, \rightarrow \rangle$ comprises a set S of states, an initial state $s_0 \in S$, a finite set Σ of labels, a transition relation $\rightarrow$ on S defined as the union of the relations $\xrightarrow{a} \subseteq S \times S$ for each a in $\Sigma \cup \{\epsilon\}$.

The relation $\rightsquigarrow$ is the counterpart for sequences in Σ^* :

- $s \overset{\varepsilon}{\rightsquigarrow} s$
- $s \overset{aw}{\rightsquigarrow} s''$ for a in $\Sigma \cup \{\epsilon\}$ and w in Σ^* if there exists s' in S such that $s \xrightarrow{a} s'$ and $s' \overset{w}{\rightsquigarrow} s''$.

We write $\mathcal{S}(s)$ for the same LTS with s in S as initial state (instead of s_0).

A *well-structured transition system* (WSTS) $\mathcal{S} = \langle S, s_0, \Sigma, \rightarrow, \leq, F \rangle$ is a labeled transition system $\langle S, s_0, \Sigma, \rightarrow \rangle$ endowed with a wqo $\leq$ on S and an $\leq$-upward closed set of final states F, such that $\rightarrow$ is *compatible* with $\leq$.

There is different definitions of the compatibility between $\rightarrow$ and $\leq$, that are not equivalent and yield different decidability results. We distinguish two restrictions of compatibility :

- $\rightarrow$ is compatible with $\leq$ if :

$$\begin{cases} s_1 \leq s_2 \\ s_1 \overset{a}{\rightsquigarrow} s_3 \end{cases} \implies \exists s_4 \geq s_3.\ s_2 \overset{a}{\rightsquigarrow} s_4$$

- $\rightarrow$ is strictly compatible with $\leq$ if it is compatible and moreover :

$$\begin{cases} s_1 < s_2 \\ s_1 \overset{a}{\rightsquigarrow} s_3 \end{cases} \implies \exists s_4 > s_3.\ s_2 \overset{a}{\rightsquigarrow} s_4$$

When we don't precise which compatibility while speaking of a WSTS, it is to be assumed that we use the first version of compatibility

3.2.1 Decidability properties

Definition 3.2.1. A WSTS is *effective* if :

- $\rightarrow$ and $\leq$ are decidable.
- For any $s \in States(\mathcal{S})$ and $a \in \Sigma \cup \{\varepsilon\}$, a finite basis of $\uparrow\mathsf{Pred}_{\mathcal{S}}(\uparrow s, a) = \uparrow\{s' \in S \mid \exists s'' \in S, s' \xrightarrow{a} s'' \text{ and } s \leq s''\}$ can effectively be computed.

The *cover set* of a WSTS is $\mathsf{Cover}_{\mathcal{S}}(s_0) = \downarrow Post^*(\downarrow s_0)$, and *Coverability* (whether a given state s belongs to $\mathsf{Cover}_{\mathcal{S}}(s_0)$) is decidable for finite branching effective WSTS, thanks to a backward algorithm that checks whether s_0 belongs to $\uparrow \mathsf{Pred}^*_{\mathcal{S}}(\uparrow s) = \uparrow\{s' \in S \mid \exists s'' \in S, s' \to^* s'' \text{ and } s'' \geq s\}$:

Proposition 3.2.2. *([5], Proposition 3.5) If $\mathcal{S}$ is an effective WSTS and B a finite set, computing a finite basis of $\uparrow Pred^*(\uparrow B)$ is effective.*

Corollary 3.2.3. *([5], Proposition 3.6) Coverability is decidable for effective WSTS.*

The following decidability results also hold :

Proposition 3.2.4. *([5], Theorem 4.6 and Theorem 4.11)*

- *Termination is decidable for effective WSTS.*
- *Boundedness is decidable for effective WSTS with strict compatibility.*

3.2.2 Bisimilarity and Weak bisimilarity

A *bisimulation relation* ([22]) $\simeq$ is an equivalence relation between LTS that has the following property :

$$\left\{\begin{array}{l} \mathcal{S}(s) \simeq \mathcal{S}'(s') \\ s \xrightarrow{a}_{\mathcal{S}} t \end{array}\right. \Rightarrow \exists t'. \left\{\begin{array}{l} \mathcal{S}(t) \simeq \mathcal{S}'(t') \\ s' \xrightarrow{a} t' \end{array}\right.$$

This notion can be weakened by using the $\rightsquigarrow$ relation instead of $\rightarrow$: A *weak bisimulation relation* ([23]) $\sim$ is an equivalence relation between LTS with the property :

$$\left\{\begin{array}{l} \mathcal{S}(s) \sim \mathcal{S}'(s') \\ s \overset{a}{\rightsquigarrow}_{\mathcal{S}} t \end{array}\right. \Rightarrow \exists t'. \left\{\begin{array}{l} \mathcal{S}(t) \sim \mathcal{S}'(t') \\ s' \overset{a}{\rightsquigarrow} t' \end{array}\right.$$

Two LTS $\mathcal{S}$ and $\mathcal{S}'$ are (resp. weakly) *bisimilar* if $\mathcal{S} \simeq \mathcal{S}'$ for some (resp. weak) bisimulation relation.

3.2.3 Weak bisimilarity in WSTS

To relate the different WSTS, we will need a notion of equivalence, that should at least preserve coverability. Indeed, the classic notion of weak bisimilarity isn't quite good to describe equivalence between WSTS. Indeed, consider two incomparable states s and s' with $s \xrightarrow{\varepsilon} s'$. Then, if we ask ourselves whether s can be covered by $\mathcal{S}(s)$, the answer is obviously yes. But (under some conditions) $\mathcal{S}(s')$ is weakly bisimilar to $\mathcal{S}(s)$, and $\mathcal{S}(s')$ will in general be unable to cover s. Thus, we define some additional notions to capture how ϵ-transitions are used :

Definition 3.2.5.

- A LTS is ε-free if there is no transitions labelled by ε.
- A LTS is ε-finite if there is no infinite computation using only ε-transitions.
- A LTS is ε-bounded if there exists a $N \in \mathbb{N}$ such that every computation using only ε-transitions is of length lower than N.

We will usually want to make ϵ-transitions truly "invisible", and this means that the order should not be able to discriminate between two states linked by ϵ-transitions :

Definition 3.2.6. A WSTS $\mathcal{S}$ is ε-coherent if :

$$s \overset{\varepsilon}{\rightsquigarrow} s' \iff s \preceq s' \land s' \preceq s$$

But this isn't enough to make bisimilarity preserve coverability. Indeed, even with ε-free systems, bisimilarity will generally not preserve this property. Assume that you have one well-structured transition systems, with one initial state s_i and two states s_a and s_b. There is only one transition from s_i to s_a. All states are incomparable. If you make s_a and s_b comparable, this wouldn't change the fact that the system is still well-structured, because no transitions are issued from these states. Thus, you get two bisimilar WSTS that have a different answer to "is s_b coverable from s_i?"

Thus, we will define a new version of equivalence, which is a stronger version of weak bisimilarity : order-preserving weak bisimilarity (shortly : *op-weak bisimilarity*)

Definition 3.2.7. Let $\mathcal{S}$ and $\mathcal{S}'$ be two ε-coherent WSTS. Let $\sim$ be a weak bisimulation. $\sim$ is an op-weak bisimulation if :

$$\begin{cases} \mathcal{S}(s_1) \sim \mathcal{S}'(s_1') \\ \mathcal{S}(s_2) \sim \mathcal{S}'(s_2') \\ s_1 \leq s_2 \end{cases} \implies s_1' \leq s_2'$$

Note that op-weak bisimulation can only be defined for ε-coherent WSTS. Indeed, as we have $s \leq s$, if $\mathcal{S}(s) \sim \mathcal{S}(s')$ (which is the case if $s \xrightarrow{\varepsilon} s'$), then we have $s \leq s' \land s' \leq s$.

The interest of this notion is that we preserve coverability (the fundamental property in WSTS) through op-weak bisimilarity :

Proposition 3.2.8. *Let $\mathcal{S}$ and $\mathcal{S}'$ be two ε-coherent WSTS. If $\mathcal{S}$ is op-weakly bisimilar to $\mathcal{S}'$ and $\mathcal{S}(s')$ is op-weakly bisimilar to $\mathcal{S}'(t')$, then $s' \in Cover(\mathcal{S}) \iff t' \in Cover(\mathcal{S}')$.*

The proof of this simple result is postponed to the appendix.

3.3 Petri Nets and Extensions

3.3.1 Vector Addition System

Petri Nets have a few equivalent definitions. The original one was introduced in [19]. Here, we define them as Vector Addition Systems. [15].

Definition 3.3.1. A Vector Addition System (VAS) of dimension p is defined by a set (V, v_0) where V is a finite subset of $\mathbb{Z}^p$ and $v_0 \in \mathbb{N}^p$.

It induces a transition system where states are elements of $\mathbb{N}^p$, the initial state is v_0, and :

$$s \to s' \iff \exists v \in V.\, s' = s + v$$

Petri Nets (or VAS) are effective well-structured systems with respect to pointwise vector comparison. Because the transition relation has strict compatibility with $\leq$, coverability and boundedness are decidable in Petri Nets. Actually, reachability is even decidable, but this is a separate result [15] that is not due to well-structure.

Petri Nets are usually represented as a graph with *p places* able to store tokens, and some number of *transitions* with incoming (linking places to transitions) and outcoming arcs (linking transitions to places) that correspond to each vector of V, showing how *firing* a transition consumes or produces tokens from places.

To fire a transition t, for each incoming arc of t, there must be a token in the associated place. Then, the effect of a transition is to remove one token for each input arc in the associated place and to add one token in each place pointed by an output arc.

As an example, figure 3.1 show a representation of the VAS :

$$\begin{aligned} V &= \{(-1,-1,1,0),(1,0,-1,1),(0,1,1,-1)\} \\ v_0 &= (2,1,1,0) \end{aligned}$$

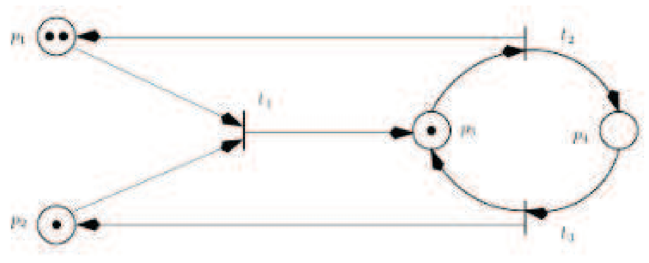

Figure 3.1: A Petri Net

3.3.2 Extensions

Simple extensions of Petri Nets are usually obtained by adding new kind of ”arcs” on the previous formalism with different semantics. Classic extensions are :

- Reset Petri Nets [10] have arcs that empty a place when the associated transition is fired.
- Transfer Petri Nets [10] have arcs that move all tokens from a place to another when the associated transition is fired. Copy arcs are a variant where the tokens from a place are duplicated into another place.
- Zero-test Petri Nets have inhibitory arcs that prevent a transition to be fired when the associated place is non-empty.

In all these models, reachability is no longer decidable. We have the following properties relating to well-structure :

- Reset Petri Nets are effective WSTS. Coverability and termination are decidable. [9]
- Transfer Petri Nets are effective WSTS with strict compatibility. Coverability, termination and boundedness are decidable.
- Zero-test Petri Nets are not effective WSTS and are actually Turing-complete.

3.3.3 Affine nets

As a Petri nets (or VAS) can be seen as a set of functions of the form $x \to x + c$ that can be applied to a state to perform a transition, Affine nets can be seen as a set of affine functions. As a non-decreasing affine function f with an upward-closed domain (and we only want to consider these functions to have compatibility with the order) can always be represented by two vectors and one matrix of non-negative integers, we define affine nets in the following way :

Definition 3.3.2. [2] An affine net $\mathcal{S}$ of dimension p is defined by (T, F, G, H), where T is a set of transitions labels, F and H are sets of vectors of $\mathbb{N}^p$ indexed by elements of T, and G is a set of matrices of $\mathbb{N}^{p\times p}$ also indexed by elements of T.

The associated LTS of initial state s_0 is $(\mathbb{N}^p, s_0, T, \to)$ where :

$$s \xrightarrow{t} s' \Leftrightarrow \begin{cases} s \geq F_t \\ s' = G_t(s - F_t) + H_t \end{cases}$$

Affine nets are a generalization of many extensions of Petri nets, including reset petri nets, transfert petri nets... These extensions can be defined by adding constraints to the G_t matrices defining the functions.

For example, standard Petri nets require $G_t = Id$, reset Petri nets that this matrix is diagonal with only zero and one eigen-values, transfert Petri nets that the matrix contains only zero and one and so on ...

A result from [2] shows that affine nets are effective well-structured transition systems, yielding that coverability and termination are decidable. Although, because they contain reset nets, we know that boundedness is undecidable [9].

3.4 Hierarchy of Petri Nets extensions

It is interesting to see that there is mainly two different ways of extending Petri Nets :

- Adding more powerful transitions, that can manipulate many tokens at once in different ways. The main difference is whether you allow whole-place operations : reset, transfert or even any affine functions.
- Using a more powerful state space, turning black tokens into named/marked tokens that can be compared in different ways. We can distinguish here three main complexities : black tokens (indistinguishable), colored tokens (only comparable for equality/inequality), data tokens (comparable by a linear order)

Systems using colored tokens are unordered Data nets ([17]) and ν-Petri Nets ([21]). The main system using comparable data token is Data Nets ([17]).

The strict inclusions of these different models has been an open problem for long. Recent works by Abdulla ([18]) and Rosa Vellardo ([11]) have shown a few reductions and strict inclusions.

Figure 3.2 shows a hierarchy of these models. Black arrows denote syntaxic inclusion, green arrows reductions based on covering languages (see section 7.1 for definitions of covering languages) and red dashed areas represent the classes of transition systems that are known to not be equivalent.

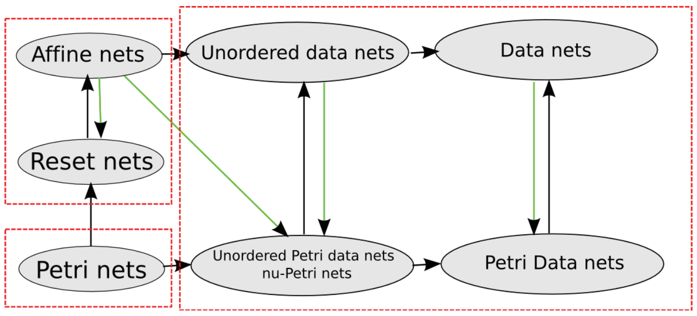

Figure 3.2: A hierarchy of Petri Net extensions

In sections 6 and 7, we will investigate a few new results regarding the separations of these extensions.

3.5 Closures and completions

If it has been known for years how to compute the upward-closure of the predecessor set $\uparrow Pred^*(\uparrow s)$ (see proposition 3.2.2), being able to compute the cover set $\downarrow Post^*(\downarrow s)$ is a more difficult problem. Indeed, we can't rely on the existence of a finite basis as the set of minimal elements. Recent works by Finkel and Goubbault ([3], [4]) have tried to give a general framework in order to compute covers of WSTS. We recall there some of their results.

The idea is that we first need to represent finitely a downward closed set and thus to *complete* the state space in a way that any downward closed set can be represented by a finite set of elements.

A *directed family* of X is any non-empty family $(x_i)_{i \in I}$ such that, for all $i, j \in I$, there is a $k \in I$ with $x_i, x_j < x_k$. A *directed-complete partially ordered set*, or *dcpo* is a partially ordered set such that every directed family has a least upper bound.

An *ideal* is a downward-closed directed family. If X is a partially ordered set, we denote by $Idl(X)$ the set of its ideals.

Proposition 3.5.1. *([7], [3]) $Idl(X)$ is a dcpo* [1] *and any downward closed subset of X can be represented as a finite union of ideals.*

$Idl(X)$ should be seen as the completion of X. $\eta : X \to Idl(X)$ that associates $\downarrow x$ to x is a canonic (order-)embedding of X in $Idl(X)$.

If $\mathbb{N}_\omega^k$ is a well-known representation of the completion of N^k, the representation of ideals more complicated structures is not immediate.

Definition 3.5.2. ([3]) Let X be a topological space.

An *atomic expressions* is either of the form A with $A \in Idl(X)$, or A^*, with A a non-empty finite subset of $Idl(X)$.

A *product* is any regular expression of the form $e_1 e_2 ... e_n$ where each e_i is an atomic expression.

Proposition 3.5.3. *The ideals of X^* and $X^\oplus$ are defined by the products on X (note that elements of multisets are defined up to permutations).*

We refer to [3] for more precision on the representation of ideals in X^* and $X^\oplus$.

[1] $Idl(X)$ is, in general, not a well-order, but it will always be the case for the structures we use (products, multisets, words). [6] describes a counterexample.

Chapter 4

Communicating Affine Nets

We introduce here a new model, that aims to have the same power as data nets, but with a more natural presentation.

4.1 Basic Communicating Affine Nets

The idea is to have a model of a set of linear-ordered communicating infinite-state processes. Moreover, we want to keep the possibility that new processes can get inserted or deleted. As a state of a process can be represented by a vector of integers, it is natural to encode a state of the whole system as a word of integer vectors of a fixed dimension p.

The expressiveness of this system depends on how each process can evolve, and how the processes can communicate between themselves. We will restrict ourselves to the study of the system when individual processes are affine nets, and consider different kind of possible transitions. For example, here are some classic transition relations :

Definition 4.1.1. A relation $\xrightarrow{Rdv} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$ is a transition relation of kind rendez-vous (shortly: Rdv) if there exists $(F_1, G_1, H_1) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ and $(F_2, G_2, H_2) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ such that :

$$u_1...u_n \to v_1...v_n \Leftrightarrow \exists i \neq j. \begin{cases} u_i \geq F_1 \\ v_i = G_1(u_1 - F_1) + H_1 \\ u_j \geq F_2 \\ v_j = G_2(u_2 - F_2) + H_2 \\ v_k = u_k \text{ for } k \neq i, j \end{cases}$$

Thus, a rendez-vous is parameterized by two affine functions. Two states are related by this transition relation iff the second can be obtained from the first when two processes perform the rendez-vous described by the two affine functions.

Definition 4.1.2. A relation $\xrightarrow{Brd} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$ is a transition relation of kind broadcast (shortly: Brd) if there exists $(F_e, G_e, H_e) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ (the emitter transition) and $(G_r, H_r) \in \mathbb{N}^{2p\times p} \times \mathbb{N}^p$ (the receivers transition) such that :

$$s \xrightarrow{t} s' \Leftrightarrow \exists i. \begin{cases} s(i) \geq F_e \\ s'(i) = G_e(s(i) - F_e) + H_e \\ s'(j) = G_r(s(i), s(j)) + H_r \end{cases}$$

It should be noted that there is only constraints on the emitter state : other processes are always able to receive a broadcast. Moreover, the receiver state is updated taking into account both the emitter state and its own state : this reflects the fact that the emitter can add information about its own state in the broadcasted message. Also, like in [14], a combination of two broadcasts can simulate a rendez-vous.

To allow us to later study many kind of transition relations, we define the transition system associated to communicating affine nets as the general form of a transitions system on $(N^p)^*$, and we will systematically precice the kind of transitions that we allow in our systems. For example, Rdv, Brd-communicating affine nets are the subclass of communicating affine nets that only communicate through rendez-vous and broadcast as described above.

There is no need to add the definitions of local transitions, as these are clearly a special case of the global transitions.

Definition 4.1.3. A *communicating affine net* of dimension p is defined by a set $\langle T, s_0, \{\xrightarrow{t}\}_{t \in T}\rangle$ where T is a set of transition labels and i $\forall t \in T.\ \xrightarrow{t} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$. This is a transition system on $(\mathbb{N}^p)^*$

It is a $Op_1, ...Op_n$-*communicating affine net* if $\forall t \in T. \exists 1 \leq j \leq n.\ \xrightarrow{t}$ is of kind Op_j.

4.2 Dynamic Creation

It is clear that if the initial state s_0 contains r processes, broadcasts will not change this number, and thus that Brd-communicating automatas is nothing more than a special case of an affine net with $r \times p$ places. This prompts the addition of transitions of kind Cre (creation), parameterized by a state s_0, that allow a new process of state t_0 to be created at any place in the array of processes.

Definition 4.2.1. A transition $\xrightarrow{t}$ is of kind creation (shortly: Cre) if there exists t_0 such that :

$$s \xrightarrow{t} s' \Leftrightarrow \exists u, v \begin{cases} s = u.v \\ s' = u.t_0.v \end{cases}$$

We have already said that Brd-communicating affine nets were equivalent to affine nets. This is not the case as soon as new processes can be created. For example, it can be mentionned that the branching degree of an affine net is bounded, while the branching degree of Cre, Brd-communicating automatas is unbounded. We show in section 6 that this gain of expressiveness implies a loss in decidability results.

4.3 Well-structure

Proposition 4.3.1. *Brd, Cre-communicating affine nets are well-structured transition systems.*

Proposition 4.3.2. *Brd, Cre-communicating affine nets have effective pred-basis.*

The proof of this simple result is available in the appendix.
This gives us the results from [5], (theorems 3.6 and 4.6) :

Proposition 4.3.3. *Coverability and Termination are decidable for Brd, Cre-communicating affine nets.*

4.4 Adding communication primitives

4.4.1 Deletion

Definition 4.4.1. A transition $\xrightarrow{t}$ is of kind deletion (shortly: Del) if there exists $\mathcal{C} \subset \mathbb{N}^p$ such that :

$$s \xrightarrow{t} s' \Leftrightarrow \exists i. \begin{cases} s(i) \in \mathcal{C} \\ s = u.s(i).v \\ s' = u.v \end{cases}$$

Unfortunately, having deletion operations in communicating automatas is of little value : If a sequence of transitions with deletions is fireable, then so is the same sequence of transitions without the deletions. Moreover, a state after a deletion is strictly lesser than the original state, meaning that if we take a system $\mathcal{S}$ and we consider $\mathcal{S}_{\setminus Del}$, the same transition system with the deletion transitions suppressed, we have :

Proposition 4.4.2.

$$\downarrow Post^*_{\mathcal{S}}(\downarrow s) = \downarrow Post^*_{\mathcal{S}_{\setminus Del}}(\downarrow s)$$

To make the deletion of processes interesting, this must be a constrained operation, meaning that we need a way to force the suppression of processes in order to perform an operation. Thus, we introduce filtering, a stronger version of deletion :

Definition 4.4.3. For u an element of $(\mathbb{N}^p)^*$, and $\mathcal{C}$ an upward closed set, we denote by $u_{|\mathcal{C}}$ the word obtained by deleting in u all letters that are not in $\mathcal{C}$.

A transition $\xrightarrow{Fil}$ is of kind filtering (shortly: Fil) if there exists $\mathcal{C} \subset \mathbb{N}^p$, $\uparrow\mathcal{C} = \mathcal{C}$ and $(F_e, G_e, H_e) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ such that :

$$us_ev \xrightarrow{t} u_{|\mathcal{C}} s'_e v_{|\mathcal{C}} \Leftrightarrow s'_e = G_e(s_e - F_e) + H_e$$

4.4.2 Reverse Broadcast

In our current model, we have a way to make a process communicate to all other process. Unfortunately, this doesn't allow for simple actions like counting the number of other alive processes. In order to be able to completely simulate data nets, we introduce another operation :

Definition 4.4.4. A relation $\xrightarrow{\overline{Brd}} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$ is a transition relation of kind reverse broadcast (shortly: $\overline{Brd}$) if there exists $(F_e, G_e, H_e) \in \mathbb{N}^p \times \mathbb{N}^{2p\times p} \times \mathbb{N}^p$ such that :

$$s \xrightarrow{t} s' \Leftrightarrow \exists i. \begin{cases} s(i) \geq F_e \\ s'(i) = G_e(s(i) - F_e, \Sigma_{i\neq j} s(j)) + H_e \end{cases}$$

Adding these two primives doesn't change our decidable properties. Indeed, the implied transitions are obviously monotic and we have :

Proposition 4.4.5. *$Brd, \overline{Brd}, Cre, Fil$-communicating affine nets have effective pred-basis.*

4.5 Adding order

All our primitives currently have no notions of adjacency, left or right. This means that states are actually multi-sets instead of words, given states are equivalent up to permutations of states. It is tempting to add these notions to the primitives, but some care must be taken :

Left and right broadcast

Definition 4.5.1. A transition $\xrightarrow{LBrd} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$ is a transition relation of kind left broadcast (shortly: $LBrd$) if there exists $(F_e, G_e, H_e) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ (the emitter transition) and $(G_r, H_r) \in \mathbb{N}^{2p\times p} \times \mathbb{N}^p$ (the receivers transition) such that :

$$u_1...u_n \xrightarrow{t} v_1...v_n \Leftrightarrow \exists i. \begin{cases} u_i \geq F_e & \\ v_i = G_e(u_i - F_e) + H_e & \\ v_j = G_r(u_i, u_j) + H_r & \text{for } j < i \\ v_j = u_j & \text{for } j > i \end{cases}$$

Right-broadcast (shortly: $RBrd$) is defined symmetrically. A transition is an ordered broadcast (shortly: $OBrd$) if it is either a left-broadcast or a right-broadcast.

All the ordered versions of the other primitives may be obtained by combining the normal version with a preliminary ordered broadcast. Thus, we will not describe the other primitives.

Ordered broadcast is a monotone operation with compatibility with $\preceq$ and it keeps effective pred-basis. Thus, the decidability results of section 4.3 still hold when adding these primitives.

Neighbor relation

Definition 4.5.2. A relation $\xrightarrow{NRdv} \in (\mathbb{N}^p)^* \times (\mathbb{N}^p)^*$ is a transition relation of kind neighbor rendez-vous (shortly: $NRdv$) if there exists $(F_1, G_1, H_1) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ and $(F_2, G_2, H_2) \in \mathbb{N}^p \times \mathbb{N}^{p\times p} \times \mathbb{N}^p$ such that :

$$u_1...u_n \rightarrow v_1...v_n \Leftrightarrow \exists 1 \leq i < n. \begin{cases} u_i \geq F_1 \\ v_i = G_1(u_i - F_1) + H_1 \\ u_{i+1} \geq F_2 \\ v_{i+1} = G_2(u_{i+1} - F_2) + H_2 \\ v_k = u_k \text{ for } k \neq i, i+1 \end{cases}$$

Proposition 4.5.3. *$NRdv$-communicating affine nets are Turing complete.*

This is a classic result for communicating finite automatas. A Turing-machine tape can be viewed as an array of communicating cells, that pass a token representing the position of the machine head.

Chapter 5

Data nets

5.1 Definition

Data nets can be seen as affine nets in which tokens have a value in a linearly-ordered, infinite domain $\mathcal{D}$. Operations of data nets can have constraints on which tokens they use based on the order.

Since the linear ordering $\leq_{\mathcal{D}}$ is the only operation available on $\mathcal{D}$, states of Petri data nets are finite sequence (words) of vectors of $\mathbb{N}^P \backslash \{0\}$. Each index j of this sequence is associated to an implicit *datum* $d_j \in \mathcal{D}$ such that $j \leq j' \iff d_j \leq d_{j'}$. For each $p \in P$, $s(j)(p)$ is the number of tokens which carry d_j and are at place p. We say that these tokens are of *kind* j. For example, the state of the data net with three places represented in figure 5.1 would be encoded as :

$$(0,1,0).(1,0,1).(0,1,2).(1,0,0)$$

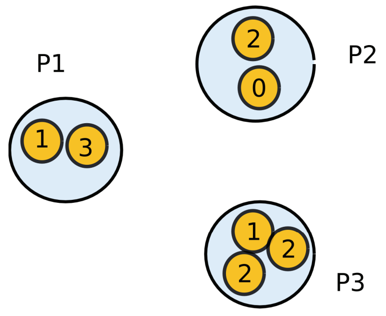

Figure 5.1: Example of a data net state

Due to the size of their definition, we will let the reader refer to [17] for the full formalism and we will only define here Petri data nets, a subset of data nets, which can be formalized more easily.

Definition 5.1.1. [17] A Petri data net is defined as $\langle P, T, \alpha, F, H\rangle$ where P is the set of places, T the set of transitions, $\alpha : T \to \mathbb{N}$ gives the *arity* of each transition. F and H are a matrixes such that for $t \in T$, $p \in P$ and $0 \leq i \leq \alpha(t)$, $F(t, p, i)$ (resp. $H(t, p, i)$) is the number of tokens of kind i that are consumed (resp. produced) in the place p when the transition t is fired.

Firing a transition t in a data net follows four steps :

- Selection of datums : $\alpha(t)$ datums are chosen nondeterministically. 0 vectors can be added in this step to the state in order to represent the choice of a fresh datum. We denote by $\iota(i)$ the i-th datum chosen. We must have $i < i' \iff \iota(i) < \iota(i')$
- Consumption : For every place p and for every $0 \leq i < \alpha(t)$, $F(t, p, \iota(i))$ tokens of kind $\iota(i)$ are removed in place p.
- Production : For every place p and for every $0 \leq i < \alpha(t)$, $H(t, p, \iota(i))$ tokens of kind $\iota(i)$ are added in place p.
- Garbage Collection : All vectors that are currently equal to 0 (meaning that their associated datum is no longer present in the net) are suppressed.

Complete data nets also support whole place operations (transfert and reset), broadcasts (very similar to the operation defined for communicating affine nets, but with the extra power of arity) and have a mechanism of collecting that can consider all tokens whose value is between two bounds. Fortunately, recent results by Abdulla *et al.* [18] have shown that Petri data nets have the full power of data nets if we allow unbounded number of ε-transitions in the weak bisimulation.

5.2 Arity in data nets

We would like to relate data nets and communicating affine nets by saying that each datum is actually a process, while interaction between datums would be communication between processes.

However, transitions of data nets are parameterized by an "arity" that, in the formalism of communicating affine nets, could be seen as the number of process initiating and controlling a transition. In real systems or protocols, it should be clear that one transition is started by only one process, and that protocols using more than one process as controllers can be simulated by the communication of the processes through broadcasts.

Without surprise, this means that data nets can be simplified to use only transitions of arity one.

Proposition 5.2.1. *Every ϵ-free data net is weakly bisimilar to an ϵ-bounded ϵ-coherent data net with only transitions of arity one.*

The idea of the proof is to make twice as many copies of the states as the maximum transition arity and to add some number of control states and boolean variables. Then, a first round of broadcasts (simulating rendez-vous), marks all the processes that will be ”controlling” the transition. A second round marks the regions, then a third communicates to each process. the sum of each region states and the controlling processes states. A fourth round makes each process compute ”locally” its resulting state. The formal proof with the technicalities is available in the appendix.

Interestingly enough, the dual property is also true : as it is possible to increase the number of transitions to have only arity one, it is possible to increase the arity in order to have only one transition. The interest of this result being limited, and the formalism being quite complicated, we refer to appendix A.4 for the formal statement.

5.3 Deletion in data nets

The major difference between communicating affine nets and data nets lie in the handling of null processes (processes that are only zeros). Although they are automatically suppressed in data nets, they stay alive in communicating affine nets, implying that they can be turned into relevant processes through broadcast. This prompts to distinguish a class of data nets :

Definition 5.3.1. A data net is non-deleting if, for all computations starting from the initial state, there is no *0-contraction*[1]

Proposition 5.3.2. *The question whether a data net is non-deleting is undecidable.*

Proof. A reset net is a special case of data net. Deciding whether the state 0 can be reached is undecidable in reset nets, as it is equivalent to reachability. □

5.4 Data nets and communicating affine nets

Once we have shown that data nets can be quasi-simulated by data nets with only transitions of arity one, we can remember that an ”ordered reverse-broadcast” can be simulated by an ordered broadcast and a reverse-broadcast, and that a transition of arity one is, up to process creation / suppression, exactly the combination of an ordered broadcast and an ordered collection.

Proposition 5.4.1. *Every data net is weakly bisimilar to a ϵ-bounded, ϵ-coherent $OBrd, \overline{Brd}, Cre, Fil$-communicating affine net. Computing this communicating affine net is effective.*

- *If the data net is unordered, the communicating affine net can be chosen to use broadcasts instead of ordered broadcasts.*
- *If the data net is non-deleting, the communicating affine net can be chosen without filtering operations.*

[1]A *0-contraction* happens when there is a null vector in the vector word obtained after a transition. This vector is deleted.

It can debated, at the light of the recent results by Abdulla ([18]), whether it is better to view Data nets as Petri data nets or as Communicating affine nets. One point for Communicating affine nets is that the reduction of data nets to this model is ϵ-bounded, while the reduction to Petri data net is only ϵ-finite. Although, it comes without discussions that Petri data nets are a lot simpler to use than Communicating affine nets.

Chapter 6

Decidability of loop acceleration

Computing the cover of Petri Nets is effective, by the Karp-Miller tree procedure [20]. It is known that as soon as we move to reset nets, computing this cover is no longer possible [9]. Works by Finkel and Goubbault ([3], [4]) have shown that with some additionnal hypothesis (that the reset net is *flattable*), computing the cover of some reset nets is possible. This result relies on *infinite-effectiveness* : the ability to compute accelerations of simple loops.

Definition 6.0.2. Let $\mathcal{S}$ be an ε-free labeled transition system and u a word of transition labels. We define $Acc_{\mathcal{S}}(s_0, u)$ the set $\downarrow\{s \mid \exists n.\ s_0 \overset{u^n}{\rightsquigarrow} s\}$.

Note that by speaking of transition labels, we mean that two different transition should be labelled by different letters : of course, if we allow all transitions to be labelled by the same letter a, computing $Acc_{\mathcal{S}}(s_0, a)$ would yield the cover.

Theorem 6.0.3. *Acceleration of simple loops is effective for affine nets.*

Proof. Because affine nets transitions are deterministic, a transition (or a sequence of transitions) is basically a partial function. With a given state, trying to iterate this transition will either stop if one of the resulting state is outside the domain of the functions, or the run will be infinite.

Deciding whether the run is infinite or finite is easy : because of well-order, by actually iterating the functions, we either end on a state that is outside the domain, or on a state that is strictly bigger than a previous one, ensuring an infinite run.

The case of a finite run being immediate, we will care only about the infinite run case :

We have $g(x) = Ax + B$, $A \in \mathbb{N}^{p*p}$, $B \in \mathbb{Z}^p$. We define $f = g^p$ for some $p \in \mathbb{N}$ such that $f(s_0) \geq s_0$. We are looking for $l = sup\{f^k(s_0) \mid k \in \mathbb{N}\}$.

We define $s_k = f^k(s_0)$ and $d_k = s_{k+1} - s_k$.

We show that $d_k = f^{k+1}(s_0) - f^k(s_0) = (As_k + B) - (As_{k-1} + B) = Ad_{k-1}$. This gives us the simple result $d_k = A^k d_0$.

Proof. We consider the following language :

$$L = \{ba^{m_1}ba^{m_2}b\dots ba^{m_k}b\sharp ba^{n_1}ba^{n_2}b\dots ba^{n_k}b \mid k \in \mathbb{N} \wedge \forall 1 \leq i \leq k.\, n_i \leq m_i\}$$

This language can be recognized easily by a data net.

However, let's consider a WSTS $\mathcal{S}$ whose state space is $(\mathbb{N}^k)^\oplus$ and assume that it recognize $\mathcal{L}$. Let σ be a sequence accepting $ba^{m_1}ba^{m_2}b\dots ba^{m_k}b\sharp ba^{m_1}ba^{m_2}b\dots ba^{m_k}b$ and let us consider a state s reached by a subsequence of σ whose image is $ba^{m_1}ba^{m_2}b\dots ba^{m_k}b$.

We define $\varphi(m_1, m_2, \dots, m_k) = s$. Then φ is a weak simulation from $(\mathbb{N}^*, \preceq)$ to $(\mathbb{N}^p)^\oplus$ by the same reasoning as in the proof of theorem 7.2.2. $\square$

More generally, any impossibility of a weak simulation between two ordered sets would probably translate to a separation between the languages that two WSTS using these state spaces can express.

Chapter 8

Conclusion and Future work

We have introduced a new model, Communicating affine nets, that are able to simulate Data nets with only bounded sequences of ε-transitions. Simultaneous works ([18]) having shown that, using unbounded sequences of ε-transitions, data nets are also simulable by Petri data nets, which are a significantly simpler model, future works should probably focus on this last model when looking at decidability results.

Thus, the hierarchy of Petri nets extensions with data seems to center around three main models : affine nets, unordered Petri data nets (or ν-Petri nets), and Petri data nets. It is interesting to note that these three models use the three classic well-ordered data structures as their state space : $\mathbb{N}^k$, $(\mathbb{N}^k)^\oplus$, $(\mathbb{N}^k)^*$.

We have shown that at least one decidability problem (acceleration of simple loops) separates affine nets from data nets. Moreover, our study of languages has shown that the languages that can be recognized by a well-structured transition system is deeply linked to the state space of that transition system. One of our result is that there exists languages of unordered Petri data nets that can not be recognized by any affine net. Extending this result to be able to differentiate unordered Petri data nets from Petri data nets seems to be an interesting open problem.

Acknowledgements I would like to thank Alain Finkel for having supervised this word, and I'm grateful to Jean Goubbault-Larrecq, Serge Haddad and Fernando Rosa Vellardo for many interesting exchanges.

Bibliography

[1] R. Mayr A. Bouajjani. Model checking lossy vector addition systems. In *STACS 99*, pages 323–333, 1999.

[2] C. Picaronny A. Finkel, P. McKenzie. A well-structured framework for analysing petri net extensions. *Information and computation*, 195:1–29, 2004.

[3] J. Goubault-Larrecq A. Finkel. Forward analysis for wsts, part i: Completions. In *STACS 2009*, pages 433–444, 2009.

[4] J. Goubault-Larrecq A. Finkel. Forward analysis for wsts, part ii: Complete wsts. In *ICALP 2009*, pages 188–199, 2009.

[5] Ph. Schnoebelen A. Finkel. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256:63–92, 2001.

[6] Parosh Aziz Abdulla and Aletta Nylén. Better is better than well: On efficient verification of infinite-state systems. In *LICS '00: Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, page 132, Washington, DC, USA, 2000. IEEE Computer Society.

[7] Samson Abramsky and Achim Jung. Domain theory. pages 1–168, 1994.

[8] Ronald V. Book and Friedrich Otto. *String-rewriting systems*. Springer-Verlag, London, UK, 1993.

[9] Ph. Schnoebelen C. Dufourd, A. Finkel. Reset nets between decidability and undecidability. In *Automata, Languages and Programming*, pages 103–115. 1998.

[10] G. Ciardo. Petri nets with marking-dependent arc cardinality: Propeties and analysis. In *15th Int. Conf. Applications and Theory of Petri Nets*, pages 179–198, 1994.

[11] G. Delzanno F. Rosa Vellardo. Language-based comparison of nets with black tokens, pure names and ordered data. In *4th International Conference on Language and Automata Theory and Applications (to appear)*, 2010.

[12] S. Purushothaman Iyer G. Cece, A. Finkel. Unreliable channels are easier to verify than perfect channels. *Information and computation*, 124:20–31, 1995.

[13] G. Higman. Ordering by divisibility in abstract algebras. In *London Math. Soc. (3)*, pages 326–336, 1952.

[14] R. Mayr J. Esparza, A. Finkel. On the verification of broadcast protocols. *Logic in Computer Science*, pages 352–359, 1999.

[15] S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 267–281, New York, NY, USA, 1982. ACM.

[16] J.B. Kruskal. The theory of well quasi-ordering: A frequently discovered concept. *Combinatorial Theory, Series A*, 13:297–305, 1972.

[17] R. Lazic. Nets with tokens which carry data. In *Petri Nets and Other Models of Concurrency*, pages 301–320. 2007.

[18] L. Van Begin P.A. Abdulla, G. Delzanno. A language-based comparison of extensions of petri nets with and without whole-place operations. In *Language and Automata Theory*

and Applications, pages 71–82. 2009.

[19] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.

[20] R.E. Miller R.M. Karp. *Parallel program schemata*. 1969.

[21] Fernando Rosa-Velardo and David de Frutos-Escrig. Name creation vs. replication in petri net systems. *Fundam. Inf.*, 88(3):329–356, 2008.

[22] R.J van Glabbeek. The linear time - branching time spectrum. In *Theories of Concurrency: Unification and Extension*, pages 278–297, 1990.

[23] R.J van Glabbeek. The linear time - branching time spectrum ii. In *Proceedings CONCUR'93*, pages 66–81, 1993.

Appendix A

Additionnal proofs

A.1 Proof of proposition 3.2.8

Proposition A.1.1. *Let $\mathcal{S}$ and $\mathcal{S}'$ be two ε-coherent WSTS. If $\mathcal{S}$ is op-weakly bisimilar to $\mathcal{S}'$ and $\mathcal{S}(s')$ is op-weakly bisimilar to $\mathcal{S}'(t')$, then $s' \in Cover(\mathcal{S}) \iff t' \in Cover(\mathcal{S}')$.*

Proof. Let s_0 and s_0' be the respective initial states of $\mathcal{S}$ and $\mathcal{S}'$ and assume that $s_0 \rightarrow^* s''$, $s'' \geq_{\mathcal{S}} s'$. Then, by a simple induction on the length of the path from s_0 to s'', there exists t'' such that $t_0 \rightarrow^* t''$, $\mathcal{S}(s'') \sim \mathcal{S}'(t'')$. Due to the order-preserving property of $\sim$, we have $t' \leq t''$, which concludes the demonstration. □

A.2 Proof of propositions 4.3.2 and 4.4.5

Lemma A.2.1. *Let ϕ be a non-decreasing affine function from $\mathbb{N}^p$ to $\mathbb{N}^q$ and y a vector of $\mathbb{N}^q$. Computing a finite basis of $\uparrow \phi^{-1}(\uparrow y)$ is effective.*

Proof. We show this by induction on p. For $p = 0$, the preimage set is obviously empty, so let's move to the inductive step.

If x is a vector of $\mathbb{N}^{p-1}$, we have $\phi(x, x_p) = \phi_0(x) + x * \phi_p$, where ϕ_0 is an affine function from $\mathbb{N}^{p-1}$ to $\mathbb{N}^q$, and ϕ_p is a vector of $\mathbb{N}^q$.

We will distinguish components of ϕ_p that are zero to the others :

$$\exists m.\ \forall 1 \leq i \leq q.\ \phi_p[i] = 0 \vee m\phi_p[i] \geq y[i]$$

This m shows a bound for x_p : Increasing x_p above m will not change the membership of $\phi(x, x_p)$ in $\uparrow y$, whatever the value of x is. Thus, we have :

$$Basis\left(\uparrow \phi^{-1}(\uparrow y)\right) = \\ \bigcup_{0 \leq k \leq m} \left\{(x, k) \mid x \in Basis\left(\uparrow \phi_0^{-1}(\uparrow (y - k\phi_p))\right)\right\}$$

□

Proposition A.2.2. *$Brd, \overline{Brd}, Cre, Fil$-communicating affine nets have effective pred-basis.*

Proof. The predecessor set of a state is the union of the predecessor sets of this state for every transition. Thus, it is enough to show that we can compute a finite basis of a predecessor set for one transition.

- $B_{\{o,u\}}(f_e, f_r)$ (Broadcast). The state s contains only a finite number of process, so we can iterate through all processes, consider the predecessor of s if one process was the sender and intersect the predecessor set obtained by lemma A.2.1 with the constraint set.
- $\overline{B}(f_e)$ (Reverse-broadcast). Same idea as broadcast.
- $F(f_e, \mathcal{C}_r)$ (Filtering). Again, we consider the predecessor sets for each possible sender. This time, for each sender, if it fulfills its constraint set, there is one unique minimal predecessor (the other predecessors being similar states, up to the insertion of non-matching processes)
- $C(s_{init})$ (Creation). States that are possible predecessors of s through a creation rule are simply those that are obtained by deletion of one process in state s_{init}.

□

A.3 Proof of proposition 5.2.1

Proposition A.3.1. *Every ϵ-free data net is weakly bisimilar to an ϵ-bounded ϵ-coherent data net with only transitions of arity one.*

Let's consider a data net $\mathcal{S} = (P, T, \alpha, F, G, H)$. We will build a data net $\mathcal{S}' = (P', T', \overline{1}, F, G, H)$, simulating $\mathcal{S}$ with only transitions of arity one.

Note: We will indistictly use "datums" (formalism of data nets) and "processes" (formalism of communicating affine nets) in this proof, referring to the same thing : a specific vector in the state.

A.3.1 Global states

The idea is that we want that at every time, we can define the *global state* of the system and that we are able, with each transition, to require that the global state was a specific one before, and that the states change with the transition.

This is really easy to do with broadcasts, and as full data nets transitions [1] can multiply every state by a given matrix, we will assume that there is an additionnal number of places, that are global control places and that at every time, there is one token for each datum in that place.

[1]as opposed to Petri data nets one

A.3.2 Splitting a transition

A transition of arity α is split in the following steps :

- α processes are marked as the emitters of the transition.
- The sum of each region states is computed.
- Each process receives a copy of each region state sum, and each emitter state.
- Each process computes his new state.

We won't describe the complete encoding of each of these operations, but simply give the main ideas for each step. Global states ensure that we always know precisely at which "step" we are.

Selection of emitters We start with a token in a random process fulfilling the constraints for the first emitter. We then set a specific place to one for all datum on the right of this emitter, and to zero on the left. Then, we select a random process fulfilling the constraints for the second emitter and that has a one in that specific place. We iterate until all emitters are chosen. We have a number of places equal to the arity of the transition with one token marking the datum associated to each emitter.

Computing sums of regions For this step, we need for every real "data place", $arity + 1$ copy places. We iterate on each emitter, from left to right, and for emitter i, we perform :

- A copy of all data places to the i-th set of copy data places for every datum greater than the emitter one.
- A deletion of all tokens with a datum greater than the emitter one in the $i - 1$-th set of copy data places.

Transferring sums of regions to all processes The first emitter can compute the sum of all tokens from each region by looking at each set of data place copies used in the previous step and can then broadcast this value to all other processes (storing it into new sets of copy places)

Then, each emitter can broadcast its own state to all process, again storing it in new sets of copy places.

Computing final state With each process containing all the required data, we iterate a last time on each emitter. For each emitter, it computes its final state and make all process on its right also compute its final state, assuming that it is a process in the region immediately on its right (if it is not, the wrong values will be overwritten when the next emitter takes hand). The first emitter also make every process on its left compute its final state.

A.3.3 Conclusion

This show that it is possible to simulate a data net with only transitions of arity one : This reduction is ϵ-bounded : if only the final step of the transition is labelled by the transition label, and all other by epsilons, the number of intermediate steps is linear in the arity of the transition and doesn't depend on any other factor. Thus, given that for each data net, we can define the maximum arity, the number of intermediate steps is bounded.

A.4 Complements to section 5.2

We have seen that we can "break" transitions of arity greater than one in simpler transitions. The reverse is also true : multiple transitions can also be merged into a single transition.

Definition A.4.1. A transition system is said to *live in domain* $\mathcal{D}$ if:

- Its initial state is in $\mathcal{D}$.
- If a state is in $\mathcal{D}$, all its successors are in $\mathcal{D}$.

Such a domain is not necessarily unique nor minimal.

Proposition A.4.2. *Any data net $\mathcal{S}$ can be effectively associated a data net $\mathcal{S}'$ living in a regular domain $\mathcal{D}$ with only one transition such that there exists a function $proj$, effective on finite set of states and on upward/downward closed set of states included in $\mathcal{D}$ with :*

$$s \rightarrow_{\mathcal{S}} s' \Rightarrow \exists (t, t') \in \mathcal{D}^2. \begin{cases} proj(t) = s \\ proj(t') = s' \\ t \rightarrow_{\mathcal{S}'} t' \end{cases}$$

$$\begin{cases} t \rightarrow_{\mathcal{S}'} t' \\ proj(t) \textit{ exists} \\ proj(t') \textit{ exists} \end{cases} \Rightarrow proj(t) \rightarrow_{\mathcal{S}} proj(t')$$

$$\forall s.\, \exists t.\, proj(t) = s$$

If $\mathcal{S}$ is non-deleting, $\mathcal{S}'$ can also be chosen non-deleting. If $\mathcal{S}$ is unordered, $\mathcal{S}'$ can also be chosen unordered.

$\mathcal{S}'$ can be seen as simulating $\mathcal{S}$ when seen through the $proj$ lens (which is effective on interesting sets). Thus, many problems of $\mathcal{S}$ can be reduced to problems of $\mathcal{S}'$. For example:

Proposition A.4.3. *For any state $s \in \mathcal{D}$ of $\mathcal{S}'$, we have :*

$$\downarrow Post^*_{\mathcal{S}}(\downarrow proj(s)) = proj(\downarrow Post^*_{\mathcal{S}'}(\downarrow s))$$

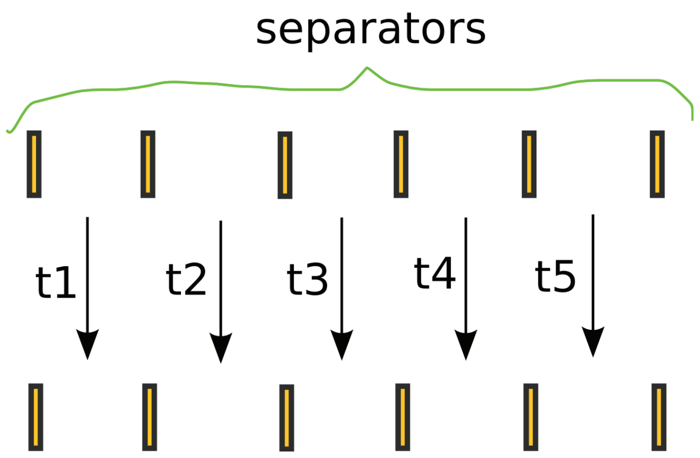

Figure A.1: An ”aggregated” transition

A.4.1 Intuition

The idea is to create a meta-transition, as shown in figure **??**, that require the presence of separators in the state, each transition being applied between two separators.

Then, if we define a state to be a *working area* and a collection of *trash areas*, all separated by separators, the transition can slide on these different areas, and when fired, it will induce the application of one specific transition to the working area. This is presented in figure A.2.

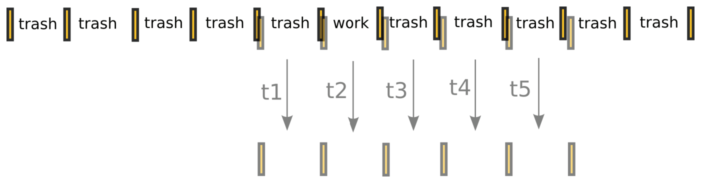

Figure A.2: Representation of the firing of a transition

Only two things must be taken care of :

- Trash areas must always contain enough data for any transition to be fired. This is simply done by repopulating these areas with huge values whenever a transition is fired.
- Contiguous separators must always be used (or only half-a-transition might be applied to the working area). Although this seems to violate the fact that we cannot use ”neighbor” relations (see proposition 4.5.3), we can cheat by deleting separators that are inside an area in which a transition is applied, and then consider all states where a separator is missing as invalid.

A.4.2 Definition of the simulating net

Let $S = (P, T, \alpha, F, G, H)$ a data net.

We will build $S' = (P \cup \{sep, work, trash\}, \{tr\}, \{\alpha_{tr}\}, \{F_{tr}\}, \{G_{tr}\}, \{H_{tr}\})$ another data net with similar semantics and only one transition.

Let $n = card(T)$, M_v the greatest value among all F_t, and M_α the greatest α_i.

We define $\{s_i\}$ to be the indices of separation between the original transitions:

$$\begin{array}{rcl} s_0 &=& 1 \\ s_{i+1} &=& s_i + \alpha_{t_i} + 1 \quad \text{for } 0 \le i \le n-1 \\ \alpha_{tr} &=& s_n \end{array}$$

Using these indices, we now define the two vectors and the matrix associated with the transition :

$$\begin{array}{rcl} F_{tr}(s_i + j, p) &=& F_{t_i}(j, p) \\ && \text{for } 0 \le i \le n-1,\ 1 \le j \le \alpha_{t_i},\ p \in P \\ F_{tr}(s_i, sep) &=& 1 \\ && \text{for } 0 \le i \le n-1 \\ F_{tr}(x, y) &=& 0 \\ && \text{otherwise} \end{array}$$

$$\begin{array}{rcl} G_{tr}(s_i + j, p, s_i + j', p') &=& G_{t_i}(j, p, j', p') \\ && \text{for } 0 \le i \le n-1,\ 1 \le j, j' \le \alpha_{t_i} \\ G_{tr}(s_i + j, p, R, p') &=& G_{t_i}(j, p, R', p') \\ && \text{for } \begin{cases} R = Reg_{(s_i+j, s_i+j+1)} \\ R' = Reg_{(j, j+1)} \\ 0 \le i \le n-1,\ 0 \le j \le \alpha_{t_i},\ p, p' \in P \end{cases} \\ G_{tr}(R, p, s_i + j, p') &=& G_{t_i}(R', p, j, p') \\ && \text{for } \begin{cases} R = Reg_{(s_i+j, s_i+j+1)} \\ R' = Reg_{(j, j+1)} \\ 0 \le i \le n-1,\ 0 \le j \le \alpha_{t_i},\ p, p' \in P \end{cases} \\ G_{tr}(R, p, R, p') &=& G_{t_i}(R', p, R', p') \\ && \text{for } \begin{cases} R = Reg_{(s_i+j, s_i+j+1)} \\ R' = Reg_{(j, j+1)} \\ 0 \le i \le n-1,\ 0 \le j \le \alpha_{t_i},\ p, p' \in P \end{cases} \end{array}$$

$$\begin{array}{rcl}
G_{tr}(s_i, trash, s_i + j, p) & = & M_v \\
& & \text{for } 0 \le i \le n-1,\ 1 \le j \le \alpha_{t_i} \\
G_{tr}(s_i, trash, R, p) & = & M_v \\
& & \text{for } \begin{cases} R = Reg_{(s_i+j, s_i+j+1)} \\ 0 \le i \le n-1,\ 0 \le j \le \alpha_{t_i},\ p \in P \end{cases}
\end{array}$$

$$\begin{array}{rcl}
G_{tr}(R, sep, R, sep) & = & 0 \\
& & \text{for } R = Reg_{(x,x+1)}, 1 \le x \le s_n \\
G_{tr}(s_i + j, sep, s_i + j, sep) & = & 0 \\
& & \text{for } 0 \le i \le n-1,\ 1 \le j \le \alpha_{t_i}
\end{array}$$

$$\begin{array}{rcl}
G_{tr}(s_i, work, s_i, work) & = & 1 \\
& & \text{for } 0 \le i \le n-1 \\
G_{tr}(s_i, trash, s_i, trash) & = & 1 \\
& & \text{for } 0 \le i \le n-1
\end{array}$$

$$\begin{array}{rcll}
H_{tr}(s_i + j, p) & = & H_{t_i}(j, p) & \text{for } 0 \le i \le n-1,\ 1 \le j \le \alpha_{t_i},\ p \in P \\
H_{tr}(R, p) & = & H_{t_i}(R', p) & \text{for } \begin{cases} R = Reg_{(s_i+j, s_i+j+1)} \\ R' = Reg_{(j,j+1)} \\ 0 \le i \le n-1,\ 0 \le j \le \alpha_{t_i},\ p \in P \end{cases} \\
H_{tr}(s_i, sep) & = & 1 & \text{for } 0 \le i \le n-1
\end{array}$$

A.4.3 Validity of configurations

A datum $d \in \mathbb{N}^{P \cup \{sep, work, trash\}}$ is said to be a separator if $d(sep) = 1$. It is a trash separator if $d(trash) = 1 \wedge d(work) = 0$ and a work separator if $d(work) = 1 \wedge d(trash) = 0$. A separator is said to be valid if $d(p) = 0$ for all $p \in P$. We write d_{trash} to represent valid trash separators, d_{work} to represent valid work separators.

Definition A.4.4. A configuration is said to be valid if all the following conditions are true :

- It contains exactly $2n$ separators
- The n-th separators is a valid work separator
- All other separators are valid trash separator
- There is no datum before the first separator
- There is no datum after the last separator
- Between any trash separator and the next separator, there is at least M_α datum whose place values are all higher than M_v.
- Non-separator datums validate $d(work) = 0, d(trash) = 0$

Definition A.4.5. A configuration is said to be invalid if it contains less than $2n$ separators. A configuration is said to be impossible if it is neither valid nor invalid.

This means a valid configuration s' of S' is of the following form :

$$d_{trash} \cdot w_1 \cdot d_{trash} \ldots d_{trash} \cdot w_{n-1} \cdot d_{work} \cdot s \cdot d_{trash} \cdot w_{n+1} \ldots \cdot d_{trash} \cdot w_{2n-1} \cdot d_{trash}$$

Here, the $\{w_i\}_{1 \leq i \leq 2n-1}$ are datum words (a list of datums) and s is a a specific datum word that is assimilated to a state of S (because s doesn't contain separators, all its non-zero components are in P). s is called the projection of s', noted $proj(s')$. This notation is extended to sets, with invalid states having no image.

Lemma A.4.6. *Only invalid configurations can be reached from invalid configurations. The set of invalid configuration is downward-closed.*

Proof. Obvious. There is no way to create new separators, so if a state has less than $2n$ separators, so have all its successors. □

Lemma A.4.7. *If, when a transition is fired, for any i, $s(i)$ and $s(i+1)$ point to non-consecutive separtors, then the resulting state is invalid.*

Proof. If two non-consecutive separators are chosen for $s(i)$ and $s(i+1)$, the separators between these two index are erased during the transition. As separators can't be created, this makes the next step invalid. □

Lemma A.4.8. *Impossible configurations can't be reached from a valid configuration. The set of impossible configuration is upward-closed.*

Proof. We need to show here that from a valid state, we can only attain valid or invalid states. So, let's take a valid state s and its successor s'. We will assume that s' is not invalid, meaning that s' has at least $2n$ separators.

Lemma gives us a part of the result. Checking the other properties of definition A.4.4 is only a matter of looking at the encoding of the simulating net.

□

This define the restrained domain $\mathcal{D}$ of the data net as the union of the valid and invalid configurations.

We remind that the completion of the data net states is the set of simple rationnal expressions whose letters are downward closed sets of $\mathbb{N}^{P \cup \{work, trash, sep\}}$. A basis of a downward closed set of states is a finite number of rationnal expressions representing the downward closed set.

Proposition A.4.9. *The basis of the projection of a downward-closed set included in $Valid \cup Invalid$ is effectively computable from the basis of the original set.*

Proof. Let $\mathcal{B}$ be a basis of a downward closed set $A \subset Valid \cup Invalid$. $A = \bigcup \{\downarrow x \mid x \in \mathcal{B}\}$

We remind that elements of $\mathcal{B}$ are of the form $u_1^{*?}u_2^{*?}u_3^{*?}...u_n^{?*}$. Let's take an element $x \in \mathcal{B}$. By definition of validity, at most $2n$ separators can be present, and none can be under the $.^*$ operator.

If less than $2n$ separators are present, then we have $\downarrow x \cap Valid = \emptyset$ and thus $proj(\downarrow x) = \emptyset$.

So if exactly $2n$ separators, then there is one element of $\downarrow x$ in $Valid$, meaning that there is exactly one work separator in the regular expression, and that we can consider the regular expression r between this separator and the next trash separator.

It is now easy to show then that $proj(\downarrow x) = r$.

□

A.4.4 Simulation

We now state that S' restricted to its valid states is in some sense simulating S.

Lemma A.4.10. *If $s_1 \rightarrow_S s_2$ and s'_1 is a valid configuration of S' with projection s_1, then there exists s'_2 a valid configuration of S' with projection s_2 such that $s'_1 \rightarrow_{S'} s'_2$*

Proof. In S, we go from s_1 to s_2 by using a transition t_p.

We will construct an explicit application of the transition relation in S'. s'_1 is a valid state, so it has $2n$ separator with the n^{th} separator being a work one. We have to choose α_{tr} datums to apply the (unique) transition to :

- For the datums of indices $\{s_i\}$ in the transition definition, we choose $n+1$ successive separators so that the p^{th} separator is the work separator.
- Between the work separator datum and the next trash separator, we have exactly s_1, so we can choose the same datum as where chosen to go from s_1 to s_2 in S (using t_p)
- Between each other separator datums, we have to choose α_{t_i} datum to apply t_i to. For all parts that are not the work part, we know by validity constraints that there is enough datum of sufficient value so that we can apply any transition of S. Thus, we can choose any of them.

We now have to show that the resulting state is valid.

Because, we have chosen consecutive separators, no separators are erased during the transition. Thus, the resulting state has exactly $2n$ separators. By proposition A.4.8, we know that we can not have reached an impossible state, so we are still in a valid state that has projection s_2

□

Lemma A.4.11. *If s_1 and s_2 are two valid configurations of S' such that $s_1 \rightarrow_{S'} s_2$, then if s'_1 is the projection of s_1 and s'_2 is the projection of s_2, we have $s'_1 \rightarrow_S s'_2$*

Proof. Because of lemma A.4.3, we know that the transition from s_1 to s_2 has chosen $n+1$ consecutive separators as datums of indices $\{s_i\}$. This means we can consider the restriction of the transition to the datums between s_p and s_{p+1}, where s_p is the work separator datum. This is exactly the application of a transition of S from $proj(s_1)$ to $proj(s_2)$.

$\square$

Lemma A.4.12. *If s is a state of S, there exists s' valid state of S' such that $proj(s') = s$*

Proof. We define $T = \{M_v\}^p \times \{O, O, O\}$. s' can then be defined by :

$$s' = d_{trash} \cdot T^{M_\alpha} \cdot d_{trash} ... d_{work} \cdot s \cdot d_{trash} \cdot T^{M_\alpha} ... d_{trash}$$

$\square$

A.4.5 Conclusion

Proposition A.4.13.

$$s \rightarrow_{\mathcal{S}} s' \Rightarrow \exists (t, t') \in \mathcal{D}^2. \begin{cases} proj(t) = s \\ proj(t') = s' \\ t \rightarrow_{\mathcal{S}'} t' \end{cases}$$

$$\begin{cases} t \rightarrow_{\mathcal{S}'} t' \\ proj(t) \text{ exists} \\ proj(t') \text{ exists} \end{cases} \Rightarrow proj(t) \rightarrow_{\mathcal{S}} proj(t')$$

$$\forall s.\ \exists t.\ proj(t) = s$$

Proof. Combination of previous results $\square$

The propositions A.4.9 and A.4.13 give us the result we were looking for.