

# Call-by-need computations in orthogonal TRSs

Irène Durand

Université Bordeaux 1

Habilitation à diriger les recherches

01/07/2005

**When**

83-84 DEA  
84-86 PHD  
86-88 Post-Doc  
89-91 MdC  
91-92 Vacataire  
92-04 MdC  
04-05 CRCT

**Where**

LSI, UPS Toulouse  
LSI, UPS Toulouse  
PRISM, Univ Maryland  
LaBRI, Univ Bordeaux 1  
Warwick Univ  
LaBRI, Univ Bordeaux 1  
FMI, Univ Stuttgart

**Programming**

Functional  
Logic  
Equational

- 1 Theoretical framework
  - Term rewriting systems
  - Rewriting strategies
  - Neededness
  - Strong Sequentiality
- 2 Theoretical contribution
  - Call-By-Need classes
  - Complexity
  - Modularity
- 3 Practical contribution
  - Autowrite
- 4 Other works
  - Computation to root-stable forms
  - Below strong sequentiality
- 5 Conclusion
- 6 Perspectives

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0))))$  ground term

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), +(s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow +(\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0))))$  reducible term  
redex

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\underbrace{\times(s(0), s(s(0)))}_{\text{redex}}) \rightarrow s(\underbrace{+( \times(0, s(s(0))), s(s(0)))}_{\text{contractum}})$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$   
 $\rightarrow s(s(s(0)))$

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$   
 $\rightarrow s(s(s(0)))$     normal form

# Term Rewriting System (TRS) $\mathcal{R}$

signature  $\mathcal{F} = \{0, s, +, \times\}$     0 constant    s unary    +  $\times$  binary

variables  $x, y, \dots$     terms  $s(s(0)), + (s(0), y)$

rewrite rules  $\mathcal{R} = \left\{ \begin{array}{l} +(0, x) \rightarrow x \\ +(s(x), y) \rightarrow s(+ (x, y)) \\ \times(0, x) \rightarrow 0 \\ \times(s(x), y) \rightarrow + (\times(x, y), y) \end{array} \right.$

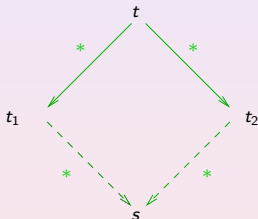
rewriting

$s(\times(s(0), s(s(0)))) \rightarrow s(+ (\times(0, s(s(0))), s(s(0))))$   
 $\rightarrow s(+ (0, s(s(0))))$   
 $\rightarrow s(s(s(0)))$  normal form

$s(\times(s(0), s(s(0)))) \rightarrow^* s(s(s(0))) \in \text{NF}(\mathcal{R})$

# Questions in Rewriting

- Is the TRS **terminating**? (no infinite rewrite sequences)
- Is the TRS **confluent**? (implies unicity of normal form)



- How to compute normal forms?

# Orthogonal Systems

## Definition

An **orthogonal** TRS is **left-linear** and **non-overlapping** (lacks critical pairs)

$$f(g(x, a)) \rightarrow x$$

$$g(a, x) \rightarrow b$$

overlapping

$$g(x, x) \rightarrow a$$

not left-linear

$$f(g(x, a)) \rightarrow g(x, x)$$

$$g(a, b) \rightarrow b$$

orthogonal

## Lemma

*orthogonality*  $\Rightarrow$  *confluence*  $\Rightarrow$  *unicity of normal form*

signature 0, fib constants s unary ;, nth, f, + binary

rewrite rules

$$\begin{array}{ll}
 +(0, y) \rightarrow y & \text{nth}(0, y : z) \rightarrow y \\
 +(s(x), y) \rightarrow s(+ (x, y)) & \text{nth}(s(x), y : z) \rightarrow \text{nth}(x, z) \\
 f(x, y) \rightarrow x : f(y, + (x, y)) & \text{fib} \rightarrow f(s(0), s(0))
 \end{array}$$

rewriting

$$\begin{array}{llll}
 \text{nth}(s(0), \text{fib}) & \rightarrow & \text{nth}(s(0), f(s(0), s(0))) & \rightarrow \\
 \text{nth}(s(0), s(0) : f(s(0), + (s(0), s(0)))) & \rightarrow & \text{nth}(0, f(s(0), + (s(0), s(0)))) & \rightarrow \\
 \text{nth}(0, f(s(0), s(+ (0, s(0)))))) & \rightarrow & \text{nth}(0, f(s(0), s(s(0)))) & \rightarrow \\
 \text{nth}(0, s(0) : f(s(s(0)), + (s(0), s(s(0)))))) & \rightarrow & s(0) & 
 \end{array}$$

$$\begin{array}{l}
 \text{nth}(s(0), \text{fib}) \rightarrow \text{nth}(s(0), f(s(0), s(0))) \rightarrow s(0) : f(s(0), + (s(0), s(0))) \rightarrow \dots \\
 \rightarrow^\omega \text{nth}(s(0), s(0) : s(0) : s^2(0) : s^3(0) : s^5(0) : \dots : \dots)
 \end{array}$$

## Definition

- **strategy** selects redexes
- strategy is **normalizing** if it computes the normal form for all terms that have one
- strategy is **sequential** if it selects a single redex

## Examples of strategies

- **leftmost outermost**      sequential
- **parallel outermost**      not sequential

## Theorem ([O'Donnell 77])

for orthogonal TRSs

- *parallel-outermost* strategy is *normalizing*
- *leftmost-outermost* strategy is *not* normalizing

$$\mathcal{R} = \begin{cases} a & \rightarrow b \\ c & \rightarrow c \\ f(x, b) & \rightarrow b \end{cases}$$

 $f(c, a) \rightarrow f(c, a) \rightarrow \dots$ 
 $f(c, a) \xrightarrow{*} f(c, b) \rightarrow b$ 

leftmost-outermost

parallel-outermost

The parallel-outermost strategy is **normalizing** but not **optimal** because it performs **useless** contractions

$$\mathcal{R} = \begin{cases} +(0, x) & \rightarrow x \\ +(s(x), y) & \rightarrow s(+ (x, y)) \\ \times(0, x) & \rightarrow 0 \\ \times(s(x), y) & \rightarrow +(\times(x, y), y) \end{cases}$$

$$\times(\underline{\times(0, s(0))}, \underline{+(0, s(0))}) \rightarrow^* \underline{\times(0, s(0))} \rightarrow 0$$

redex  $+(0, s(0))$  is not **needed**

$$\times(\underline{\times(0, s(0))}, + (0, s(0))) \rightarrow \underline{\times(0, + (0, s(0)))} \rightarrow 0$$

## Definition ([Huet & Lévy 79])

A redex  $\Delta$  in a term is **needed** if a descendant of  $\Delta$  is contracted in every rewrite sequence from this term to normal form

## Theorem ([Huet & Lévy 79])

for *orthogonal* TRSs ( $\perp$ )

- every reducible term has a needed redex
- needed rewriting gives an optimal normalizing strategy

## Definition

A strategy which contracts only **needed** redexes is called a **Call-By-Need** (CBN) strategy

# Strong Sequentiality [HL 79]

- Unfortunately: it is **undecidable** whether a redex is needed
- find **decidable** approximation of needed redex

## Definition

**strongly needed redex**: contracted in any rewrite sequence to normal form using **arbitrary** right-hand sides.

- complicated definition
- notion of **index**, **sequentiality** of predicate on term prefixes

In orthogonal systems not every reducible term has a strongly-needed redex

## Definition

**strongly sequential** systems: every reducible term has a strongly needed redex

Theorem ([Huet & Lévy 79])

*It is decidable whether a redex in a term is strongly needed*

Theorem ([Huet & Lévy 79])

*It is decidable whether an orthogonal TRS is strongly sequential.*

Proof.

proof is quite difficult (uses the notion of matching dag)

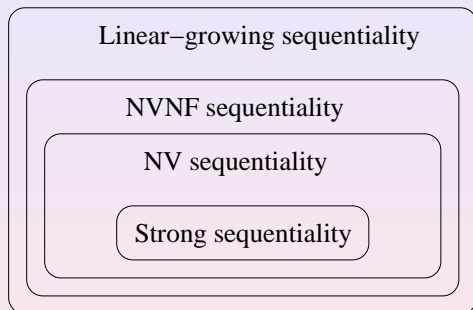


other proofs

- [Klop & Middeldorp 91] (deltaset)
- [Comon 95,00] (WSkS)

Huet and Lévy's theorem gave rise to several generalizations

# Generalization of strong sequentiality



# Theoretical contribution

One of our main contribution to the domain has been to give a **uniform** and **simplified** framework to define classes which admit decidable call-by-need strategies (joint work with Aart Middeldorp).

The benefits are

- simpler definitions
- simpler proofs
- bigger classes

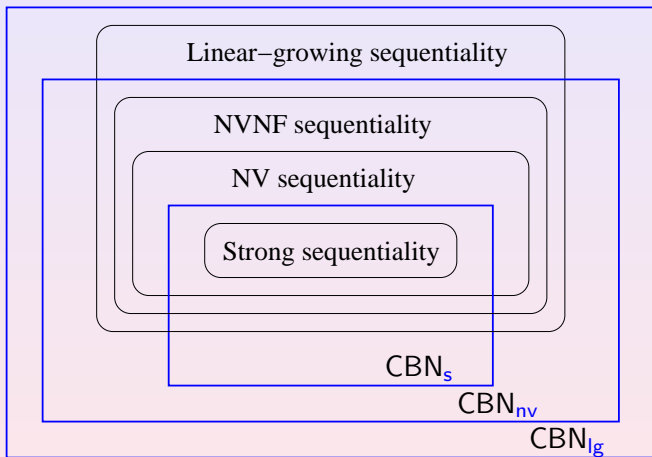
Linear-growing sequentiality

NVNF sequentiality

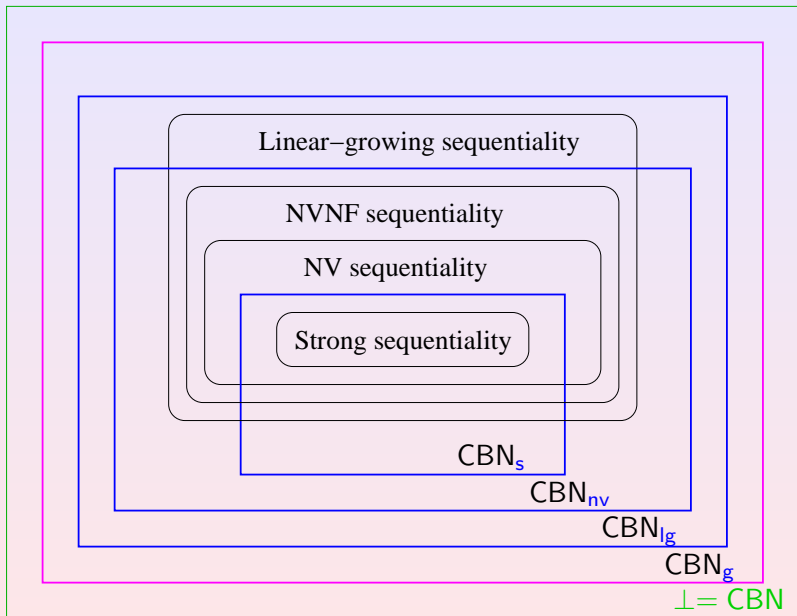
NV sequentiality

Strong sequentiality





$\perp = CBN$



Notation:  $(\xrightarrow{*}_{\mathcal{R}})[\mathcal{L}] = \{t \mid t \xrightarrow{*}_{\mathcal{R}} s \in \mathcal{L}\}$

### Lemma

for an *orthogonal* TRS  $\mathcal{R}$

*redex*  $\Delta$  in  $C[\Delta]$  needed  $\iff C[\bullet] \notin (\xrightarrow{*}_{\mathcal{R}})[\text{NF}]$      *•-free nf*

Key idea: *approximate*  $\mathcal{R}$  by TRS  $\mathcal{R}_\alpha$  such that  $C[\bullet] \notin (\xrightarrow{*}_{\mathcal{R}})[\text{NF}]$  is decidable.

### Definition

TRS  $\mathcal{R}_\alpha$  *approximates*  $\mathcal{R}$  if  $\xrightarrow{*}_{\mathcal{R}} \subseteq \xrightarrow{*}_{\mathcal{R}_\alpha}$  and  $\text{LHS}_{\mathcal{R}} = \text{LHS}_{\mathcal{R}_\alpha}$

# Approximations

- **strong** (s) [Huet & Lévy 79]  
 replace right-hand sides by fresh variables
- **non-variable** (nv) [Oyamaguchi 93]  
 replace variables in right-hand sides by fresh variables
- **linear-growing** (lg) [Jacquemard 96]  
growing (g) [Nagaya & Toyama 99]  
 replace variables in right-hand sides that occur at depth  $> 1$   
 in left-hand sides by fresh variables

$$\xrightarrow{*} \mathcal{R} \subseteq \xrightarrow{*} \mathcal{R}_g \subseteq \xrightarrow{*} \mathcal{R}_{lg} \subseteq \xrightarrow{*} \mathcal{R}_{nv} \subseteq \xrightarrow{*} \mathcal{R}_s$$

## Lemma

if  $\mathcal{R}_\alpha$  approximates an *orthogonal* TRS  $\mathcal{R}$  then  $\mathcal{R}_\alpha$ -needed redexes are  $\mathcal{R}$ -needed (= needed)

Observation: If every *reducible* term has an  $\mathcal{R}_\alpha$ -needed redex,  $\mathcal{R}$  admits an *optimal* and *computable* sequential call-by-need strategy.

## Definition ([Durand-Middeldorp 97])

The class of *orthogonal* TRSs  $\mathcal{R}$  such that every reducible term has an  $\mathcal{R}_\alpha$ -needed redex is called **CBN $_\alpha$** .

$$\text{CBN}_s \subsetneq \text{CBN}_{nv} \subsetneq \text{CBN}_{lg} \subsetneq \text{CBN}_g \subsetneq \text{CBN} = \perp$$

## Definition

TRS  $\mathcal{R}$  is **recognizability preserving** if for every recognizable set  $\mathcal{L}$   $(\xrightarrow{*}_{\mathcal{R}})[\mathcal{L}]$  is recognizable.

Theorem ([Jaquemard 96], [Dur-Mid 97], [Nagaya-Toyama 99])

For **left-linear**  $\mathcal{R}$  and  $\alpha \in \{\mathbf{s}, \mathbf{nv}, \mathbf{lg}, \mathbf{g}\}$ ,  $\mathcal{R}_\alpha$  is **recognizability preserving**.

$\Rightarrow (\xrightarrow{*}_{\mathcal{R}_\alpha})[\mathbf{NF}]$  is recognizable

$\Rightarrow$  It is decidable whether  $C[\bullet] \notin (\xrightarrow{*}_{\mathcal{R}_\alpha})[\mathbf{NF}]$

$\Rightarrow$  It is decidable whether a redex is  $\mathcal{R}_\alpha$ -needed

$$\xrightarrow{*}_{\mathcal{R}} \subseteq \xrightarrow{*}_{\mathcal{R}_g} \subseteq \xrightarrow{*}_{\mathcal{R}_{lg}} \subseteq \xrightarrow{*}_{\mathcal{R}_{nv}} \subseteq \xrightarrow{*}_{\mathcal{R}_s}$$

### Theorem ([Comon 95])

*The set of reducible terms without  $\mathcal{R}_s$ -needed redex is recognizable.*

### Theorem ([Durand-Middelorp 97])

*If  $\mathcal{R}_\alpha$  is recognizability preserving then the set of reducible terms without  $\mathcal{R}_\alpha$ -needed redex is recognizable.*

### Corollary

*If is decidable whether a left-linear TRS belongs to  $\text{CBN}_\alpha$  for  $\alpha \in \{s, nv, lg, g\}$ .*

# Results

For the  $CBN_{\alpha}$  classes, we have obtained

- decidability results
- complexity results
- modularity results

# Complexity

- $\mathcal{R} \in \text{CBN}_s ?$  exponential [Comon 95,00]
- $\mathcal{R} \in \text{CBN}_{nv} ?$  [Durand-Middeldorp 98]
- $\mathcal{R} \in \text{CBN}_{lg} ?$  double exponential
- $\mathcal{R} \in \text{CBN}_g ?$  triple exponential [Durand 05]
- $\mathcal{R} \in \text{FB} ?$  quadratic [Durand 94]  
 $\text{FB} \subsetneq \text{CBN}_s$  [Strandh 89]

# Modularity

Motivation: Since deciding membership in  $\text{CBN}_\alpha$  is complex  
**modularity results** are important

- Modularity

$$\begin{array}{l} (\mathcal{R}_1, \mathcal{F}_1) \in \text{CBN}_\alpha \\ (\mathcal{R}_2, \mathcal{F}_2) \in \text{CBN}_\alpha \end{array} \xRightarrow{?} (\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{F}_1 \cup \mathcal{F}_2) \in \text{CBN}_\alpha$$

- First step towards modularity: **Signature extension**

$$\begin{array}{l} (\mathcal{R}, \mathcal{F}) \in \text{CBN}_\alpha \\ \mathcal{F} \subsetneq \mathcal{G} \end{array} \xRightarrow{?} (\mathcal{R}, \mathcal{G}) \in \text{CBN}_\alpha$$

- Neither one of these two implications hold in general.

# Modularity

Motivation: Since deciding membership in  $\text{CBN}_\alpha$  is complex  
**modularity results** are important

- Modularity

$$\begin{array}{l} (\mathcal{R}_1, \mathcal{F}_1) \in \text{CBN}_\alpha \\ (\mathcal{R}_2, \mathcal{F}_2) \in \text{CBN}_\alpha \end{array} \xRightarrow{?} (\mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{F}_1 \cup \mathcal{F}_2) \in \text{CBN}_\alpha$$

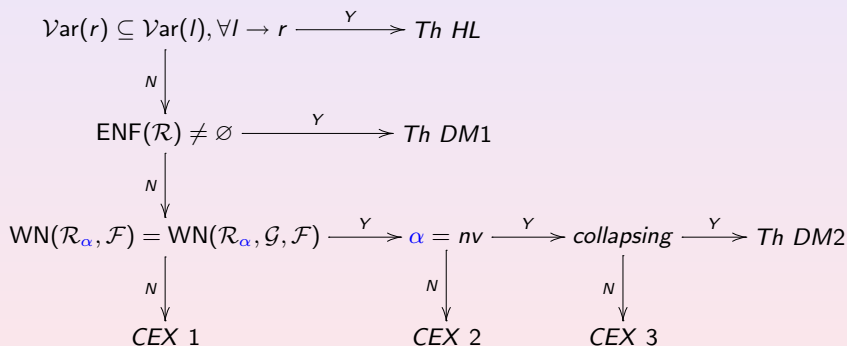
- First step towards modularity: **Signature extension**

$$\begin{array}{l} (\mathcal{R}, \mathcal{F}) \in \text{CBN}_\alpha \\ \mathcal{F} \subsetneq \mathcal{G} \end{array} \xRightarrow{?} (\mathcal{R}, \mathcal{G}) \in \text{CBN}_\alpha$$

- Neither one of these two implications hold in general.

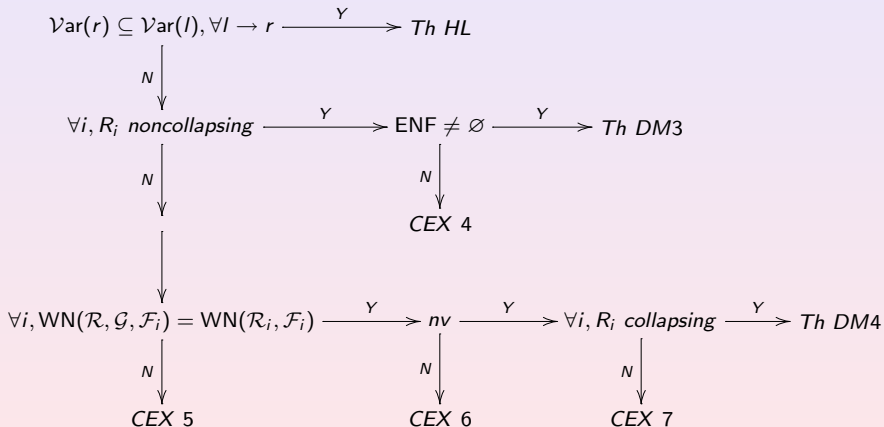
# Signature extension

Results concerning signature extension [Durand-Middeldorp 01]:



# Modularity

Results concerning modularity [Durand-Middelorp 05]:



Also results for **constructor sharing** combinations

# The need for Autowrite

useful properties for obtaining sufficient conditions:

- $NF(\mathcal{R}, \mathcal{F}) \neq \emptyset$
- $ENF(\mathcal{R}, \mathcal{F}) \neq \emptyset$
- $WN(\mathcal{R}_\alpha, \mathcal{G}, \mathcal{F}) = WN(\mathcal{R}_\alpha, \mathcal{F})$
- is  $\mathcal{R}_\alpha$  collapsing? arbitrary?

restriction  $\Rightarrow$  counterexample

For each counterexample, we needed to check that  $(\mathcal{R}, \mathcal{F}) \in \text{CBN}_\alpha$ ,  $(\mathcal{R}, \mathcal{G}) \notin \text{CBN}_\alpha$  and some of the above conditions.

$\Rightarrow$  many tedious proofs

$\Rightarrow$  Autowrite instead

# Autowrite

Main algorithms implemented in Autowrite

## Automata

- boolean operations
- emptiness problem
- emptiness of intersection

## Term Rewriting Systems

For **left-linear**  $\mathcal{R}$ ,  $\alpha \in \{\mathbf{s}, \mathbf{nv}, \mathbf{lg}, \mathbf{g}\}$  and automaton  $\mathcal{A}$ ,

- **Build** an automaton  $\mathcal{C}_{\mathcal{R}_\alpha, \mathcal{A}}$  such that
$$L(\mathcal{C}_{\mathcal{R}_\alpha, \mathcal{A}}) = \left(\frac{*}{\mathcal{R}_\alpha}\right)[L(\mathcal{A})],$$
- **Build** an automaton  $\mathcal{D}_{\mathcal{R}_\alpha}$  recognizing the set of reducible terms without  $\mathcal{R}_\alpha$ -needed redexes.

Most of the other operations are combinations of the above.

# The outside Autowrite

Autowrite handles a set of **specifications**  
each specification contains

- a **signature**,
- possibly a set of **variables**,
- a list of Autowrite **objects** built upon the signature and variables

The Autowrite objects are:

- **Term**
- **Termset** [a set of terms] (named)
- **TRS** (named)
- **Automaton** (named)

# Example of a specification

Ops 0:0 s:1 +:2 \*:2

Vars x y

TRS R

; addition

+ (0, x) → x

+ (s(x), y) → s(+ (x, y))

; product

\* (0, x) → 0

\* (s(x), y) → + (\* (x, y), y)

Automaton EVEN

States odd even

Final States even

Transitions

0 → even

s(even) → odd

s(odd) → even

Termset RS

0 s(x)

Term (\* (\* (0, s(0)), + (0, s(0)))

Term \*(o, + (0, s(0)))

Term (\* (\* (0, s(0)), o)

Term s(s(s(0)))

# The inside Autowrite

- core of the system written in **CLOS** (Common Lisp Oriented System)
- the graphical interface is written using **McCLIM** the free implementation of the CLIM specification
- altogether about **7500** lines of code

## Choices for better performance:

- use **sharing** rather than copying (especially for automata states)
- **compare references** not the contents of the objects
- use **hash tables**
- use **memoization**

- Autowrite has been used to check many of our examples
- Autowrite has helped us finding counterexamples
- Easy to install (self-contained) and use (no Lisp knowlegde required)
- Autowrite runs faster than other systems implementing tree automata like **Timbuk** or **RX**
- **Available** from my Web page:  
<http://dept-info.u-bordeaux.fr/~idurand>

## Perspectives

- improve performance
- apply to termination
- extend to other types of automata

# Computation to root-stable forms

## Definition

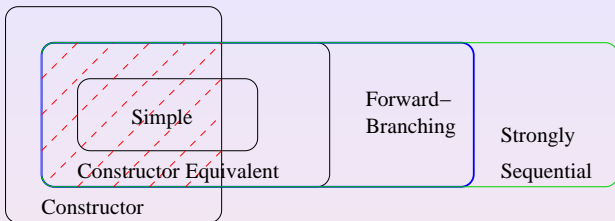
a term is **root-stable** if it does not rewrite to a redex  
(the useful notion for dealing with **infinite** normal forms)

- [Middeldorp 97] shows that needed redexes are not adequate for computing root-stable forms and proposes the notion **root-needed** redex
- difficulties for extending our framework to compute root-stable forms
  - **undecidability** of root-stability
  - root-neededness depends not only the position but also on the redex itself

## Results:

- **definition** of CBN-RS $_{\alpha,\beta}$  classes with decidable call-by-need strategy to compute  $\alpha$ -root-stable forms
- **decidability** and **complexity** results

# Below strong sequentiality



- quadratic decision algorithm for Forward-Branching systems [Durand 94]
- Bruno Salinier PHD's thesis
  - definition of the class of Constructor Equivalent TRSs [Durand-Salinier 93,94]
  - transformation from forward-branching to strong sequential constructor [Salinier-Strandh 96,97]
- implementation of the transformation in Autowrite [Durand 04]

# Strong sequentiality

Conjecture ([Middeldorp and Klop 91])

*Deciding strong sequentiality is NP-complete  
(problem #9 in the list of RTA open problems)*

...

more than 12 years of research

...

Conjecture ([Durand 05])

*the problem is both in NP and co-NP*

# Conclusion

—

- Results on computations to root-stable forms still unpublished
- No result on Klop and Middeldorp's conjecture

+

- Theoretical and practical contribution
- Bruno Salinier PHD's thesis
- Journal articles: [IPL 93], [IPL 94], [JSC 94]  
[ENTCS 05] [IC 05]
- Conferences communications: [PEPM 91],  
[CADE 97], [FOSSACS 01], [RTA 02], [WRS 04]

# Conclusion

—

- Results on computations to root-stable forms still unpublished
- No result on Klop and Middeldorp's conjecture

+

- Theoretical and practical contribution
- Bruno Salinier PHD's thesis
- Journal articles: [IPL 93], [IPL 94], [JSC 94]  
[ENTCS 05] [IC 05]
- Conferences communications: [PEPM 91],  
[CADE 97], [FOSSACS 01], [RTA 02], [WRS 04]

# Perspectives

- generalize systems that preserve recognizability
- apply automata techniques to termination
- computational linguistics
- ?