



Modular Termination of Context-Sensitive Rewriting

Salvador Lucas
Universidad Politécnica de Valencia
`slucas@dsic.upv.es`

joint work with

Bernhard Gramlich
Technische Universität Wien
`gramlich@logic.at`

Motivation: programming languages

- **User-defined strategies** in rewriting-based programming/specification languages (e.g., ELAN, Maude, OBJ2, OBJ3, CafeOBJ, Stratego,...)
- Often used together with (more) standard features; for instance: **Modules**

Motivation: programming languages

Example: Consider the following OBJ program:

```
obj EXAMPLE is
  sorts Nat LNat .
  op 0      : -> Nat .
  op s      : Nat -> Nat [strat (0)] .
  op nil    : -> LNat .
  op cons   : Nat LNat -> LNat [strat (1)] .
  op inf    : Nat -> LNat [strat (1 0)] .
  op take   : Nat LNat -> Nat [strat (1 2 0)] .
  op length : LNat -> Nat [strat (1 0)] .
  vars N X  : Nat .
  var L     : LNat .
  eq inf(N) = cons(N,inf(s(N))) .
  eq take(0,X) = nil .
  eq take(s(N),cons(X,L)) = cons(X,take(N,L)) .
  eq length(nil) = 0 .
  eq length(cons(X,L)) = s(length(L)) .
endo
```

Explicit local
strategies

Missing arguments are
not evaluated!



Motivation: programming languages

*How to analyze the computational properties
of programs written in **such** languages?*

Motivation: programming languages

*Termination of rewriting and termination
of such programs do **not** coincide!!*

Motivation: programming languages

Example:

```
sel(s(0), from(0))      sel(s(0), cons(0, from(s(0))))
```

Terminating!?

```
sel(s(0), from(0))  
sel(s(0), cons(0, cons(s(0), from(s(s(0)))))  
sel(s(0), cons(0, s(0), s(s(0)), from(s(s(s(0))))))  
...
```

Impossible (due to
replacement restrictions)!

Motivation: programming languages

Example: Consider the following program:

```
obj EXAMPLE is
  sorts Nat LNat .
  op 0
  op s
  op n
  op c
  op i
  op t
  op l
  vars N X      : Nat .
  var L         : LNat .
  eq inf(N) = cons(N,inf(s(N))) .
  eq take(0,X) = nil .
  eq take(s(N),cons(X,L)) = cons(X,take(N,L)) .
  eq length(nil) = 0 .
  eq length(cons(X,L)) = s(length(L)) .
endo
```

Yes, and provable!

Motivation: programming languages

Example: Consider the following program:

```
obj EXAMPLE is
  sorts Nat LNat .
```

Key: Syntactic replacement restrictions /
Context-sensitive rewriting

```
vars N X      : Nat .
var L         : LNat .
eq inf(N) = cons(N,inf(s(N))) .
eq take(0,X) = nil .
eq take(s(N),cons(X,L)) = cons(X,take(N,L)) .
eq length(nil) = 0 .
eq length(cons(X,L)) = s(length(L)) .
endo
```

Motivation: programming languages

- Replacement maps:

sets (rather than lists) of positive integers associated to function symbols aimed at specifying which arguments are *reducible* in function calls

- Context-Sensitive Rewriting (CSR [JFLP98]):

rewriting steps are limited by the replacement restrictions imposed by a replacement map

- Useful for modeling local strategies [PPDP'01,LPAR'01]

Motivation: programming languages

Example

```
obj EXAMPLE is
  sorts Nat LNat .
  op 0      : => Nat
```

Sig

```
{0, s, nil, cons, inf,
 take, length}
```

Replacement
map

```
 $\mu(s)=$   
 $\mu(\text{cons})=\{1\}$   
 $\mu(\text{inf})=\{1\}$   
 $\mu(\text{take})=\{1,2\}$   
 $\mu(\text{length})=\{1\}$ 
```

TRS

```
inf(N) -> cons(N, inf(s(N)))
take(0, L) -> nil
take(s(N), cons(X, L)) -> cons(X, take(N, L))
length(nil) -> 0
length(cons(X, L)) -> s(length(L))
```

```
endo
```

Motivation: programming languages

- Several methods for proving termination of CSR have been developed by:
 - ◆ Lucas [ICALP'96]
 - ◆ Zantema [RTA'97]
 - ◆ Ferreira and Ribeiro [RTA'99]
 - ◆ Giesl and Middeldorp [RTA'99]
 - ◆ Borralleras, Lucas, and Rubio [CADE'02]
 - ◆ Gramlich and Lucas [RULE'02]
- Other approaches: CARIBOO [FGK02]

Motivation: modular programs

■ Programs often use libraries of functions

Example: The syntactic equality of expressions of sort `Nat` can be tested via the OBJ 'built-in' operator '===', which can be modeled as follows:

```
obj ID-NAT is
  op 0          : -> Nat .
  op s          : Nat -> Nat .
  op _===_      : Nat Nat -> Bool [strat (0)] .
  vars M N     : Nat .
  eq 0 === 0 = true .
  eq s(M) === s(N) = M === N .
  eq M === N = false .
endo
```

Motivation: modular programs

What about using **EXAMPLE**
and **ID-NAT** **together**?

Is the resulting program
terminating?

Motivation: modular programs

*Investigating modular termination of CSR
is useful for answering this question!*



Summary

- Modularity of termination
- Context-sensitive rewriting (CSR)
- Termination of CSR
- Modularity of termination of CSR
- Conclusions and future work



Modularity of termination

Modularity of termination

- Given TRSs \mathbf{R} and \mathbf{S} , we want to know whether termination of both \mathbf{R} and \mathbf{S} imply termination of $\mathbf{R} + \mathbf{S}$
- In general, termination is **not** a modular property of TRSs! Toyama's example:

$\mathbf{R}: f(a, b, x) \rightarrow f(x, x, x)$

terminating

$\mathbf{S}: g(x, y) \rightarrow x$
 $g(x, y) \rightarrow y$

terminating

Modularity of termination

- The union $\mathbf{R} + \mathbf{S}$ of both (disjoint) systems is **not** terminating:

$f(\underline{g(a,b)}, g(a,b), g(a,b))$

$\rightarrow f(a, \underline{g(a,b)}, g(a,b))$

$\rightarrow \underline{f(a,b,g(a,b))}$

$\rightarrow f(\underline{g(a,b)}, g(a,b), g(a,b))$

$\rightarrow \dots$

Modularity of termination

- Some positive results have been envisaged when the modules are:

◆ **disjoint**: they do not share any symbol

◆ **constructor-sharing**: shared symbols are constructor symbols in both modules

◆ **composable**: shared defined symbols must be defined by the same rules in both modules

- Modularity is easier to achieve for weaker termination properties (e.g., normalization, innermost termination, ...)



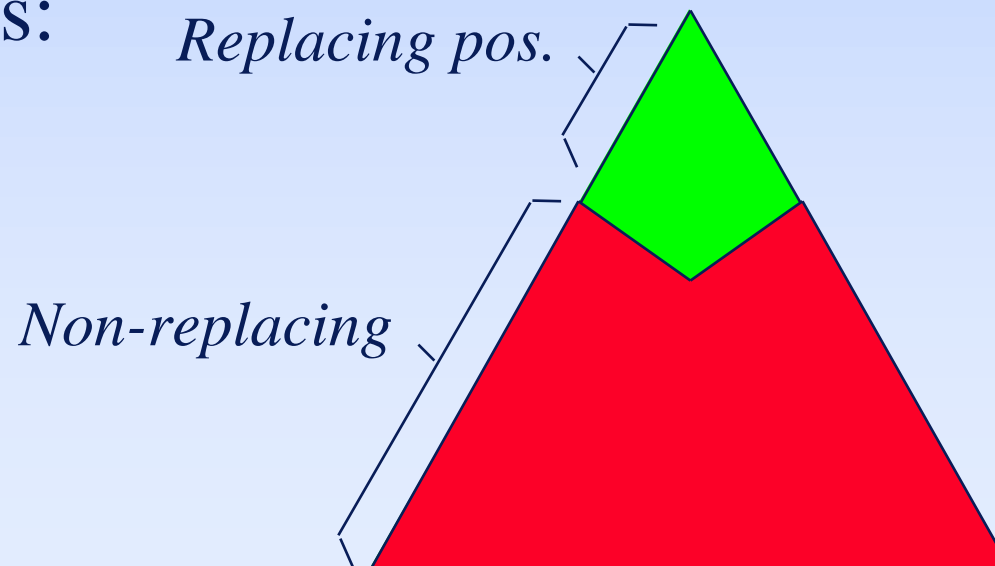
Context-sensitive rewriting

Context-sensitive rewriting

■ Replacement map:

A mapping μ which associates a subset $\mu(f)$ of $\{1, \dots, ar(f)\}$ to each function symbol f

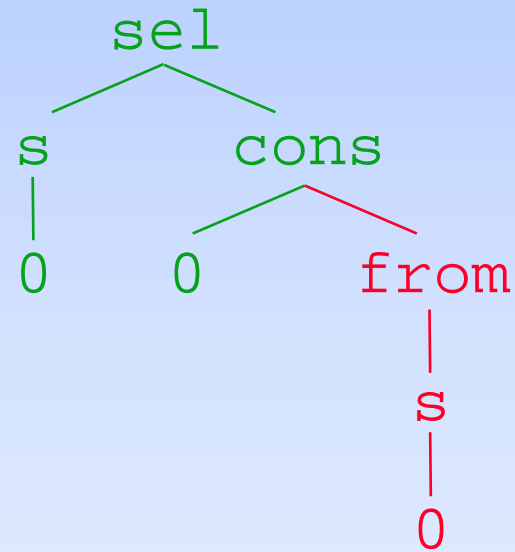
■ The set of positions of a term splits into two disjoint sets:



Context-sensitive rewriting

■ Example: $t = \text{sel}(s(0), \text{cons}(0, \text{from}(s(0))))$

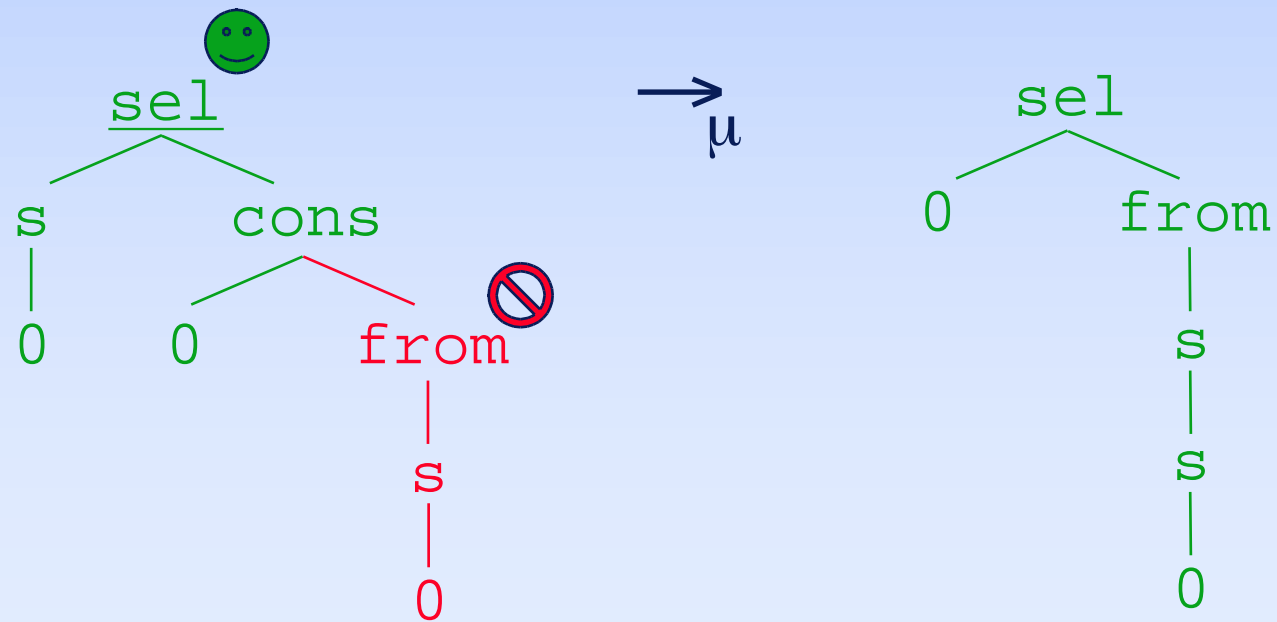
$\mu(0) =$
$\mu(s) = \{1\}$
$\mu(\text{nil}) =$
$\mu(\text{cons}) = \{1\}$
$\mu(\text{from}) = \{1\}$
$\mu(\text{sel}) = \{1, 2\}$



Context-sensitive rewriting

■ Context-sensitive rewriting:

Given a replacement map μ , *only* rewriting steps at replacing positions are allowed (written $t \rightarrow_{\mu} s$)



Context-sensitive rewriting

- **Definition:** A **CSRS** is a pair (R, μ) , where R is a TRS and μ is a replacement map



Termination of CSR

Termination of CSR

- **Transformations:** Termination of a CSRS (R, μ) is demonstrated by proving termination of a TRS R^{μ}_{\ominus} for a given transformation :
 - ◆ Lucas [ICALP96]
 - ◆ Zantema [RTA97]
 - ◆ Ferreira and Ribeiro [RTA99]
 - ◆ Giesl and Middeldorp [RTA99] (gave a **complete** transformation)

Termination of CSR

- Lucas' transformation [ICALP96]: **remove** all non-replacing subterms from the rules of the TRS R : For instance:

$$R \left\{ \begin{array}{l} \text{first}(0, x) \rightarrow \text{nil} \\ \text{first}(s(x), \text{cons}(y, z)) \rightarrow \text{cons}(y, \text{first}(x, z)) \\ \text{from}(x) \rightarrow \text{cons}(x, \text{from}(s(x))) \end{array} \right.$$

$$\mu(\text{cons}) = \mu(\text{from}) = \mu(s) = \{1\}, \mu(\text{first}) = \{1, 2\}$$

$$R_L^\mu \left\{ \begin{array}{l} \text{first}(0, x) \rightarrow \text{nil} \\ \text{first}(s(x), \text{cons}(y)) \rightarrow \text{cons}(y) \\ \text{from}(x) \rightarrow \text{cons}(x) \end{array} \right.$$

Note that R_L^μ is terminating (e.g., use *RPO*)

Termination of CSR

- Zantema's transformation [RTA97]: the non-replacing subterms of the rules are **marked**:

$$R \left\{ \begin{array}{l} \text{sel}(s(N), \text{cons}(X, L)) \rightarrow \text{sel}(N, L) \\ \text{sel}(0, \text{cons}(X, L)) \rightarrow X \\ \text{from}(N) \rightarrow \text{cons}(N, \text{from}(s(N))) \end{array} \right.$$

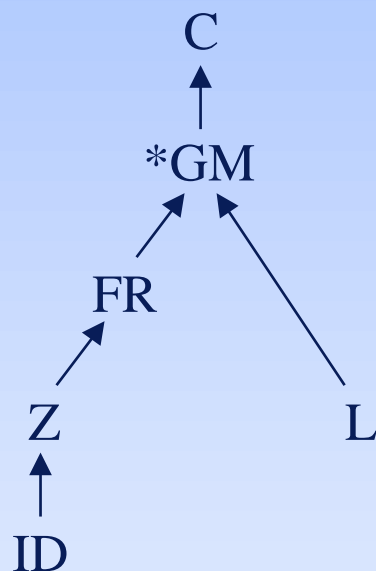
$\mu(\text{cons}) = \mu(\text{from}) = \mu(s) = \{1\}, \mu(\text{sel}) = \{1, 2\}$

$$R_Z^\mu \left\{ \begin{array}{l} \text{sel}(s(N), \text{cons}(X, L)) \rightarrow \text{sel}(N, \text{activate}(L)) \\ \text{sel}(0, \text{cons}(X, L)) \rightarrow X \\ \text{from}(N) \rightarrow \text{cons}(N, \text{from}'(s(N))) \\ \text{from}(N) \rightarrow \text{from}'(N) \\ \text{activate}(\text{from}'(N)) \rightarrow \text{from}(N) \\ \text{activate}(X) \rightarrow X \end{array} \right.$$

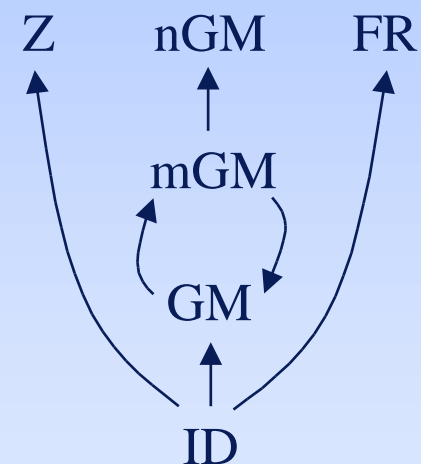
Note that R_Z^μ is terminating (use *RPO* again)

Termination of CSR

■ Comparison:



Termination [GM99]



Simple term. [RTA02]

The transformations are available for use within MU-TERM 1.0:

<http://www.dsic.upv.es/users/elp/slucas/muterm>

Termination of CSR

Are *indirect* modular proofs of termination of CSR possible?

No specific modularity analysis for CSR would be needed!

Termination of CSR

- **Idea:** Given CSRSs (\mathbf{R}, μ) and (\mathbf{S}, μ') which satisfy some combinatory criterion M for modularity (e.g., disjointness, constructor-sharingness, ...), we:
 - ◆ **transform** them using a transformation τ , next
 - ◆ **prove termination** of both $\mathbf{R}^{\mu}_{\Theta}$ and $\mathbf{S}^{\mu'}_{\Theta}$; finally
 - ◆ try to **use a combinatory criterion M'** (maybe M itself) for ensuring termination of $\mathbf{R}^{\mu}_{\Theta} + \mathbf{S}^{\mu'}_{\Theta}$, hence of $(\mathbf{R}, \mu) + (\mathbf{S}, \mu')$

Termination of CSR

- **Good point:** Such a procedure works fine for Lucas' transformation (which does not introduce new symbols and rules)
- **Not so good:** The previous approach does **not** work with Zantema's, Ferreira and Ribeiro's, and Giesl and Middeldorp's transformation!

The problem is that such transformations introduce new shared symbols and rules defining them!

Termination of CSR

■ Example: Zantema's transformation:

$$R \left\{ \begin{array}{l} \text{sel}(s(N), \text{cons}(X, L)) \rightarrow \text{sel}(N, L) \\ \text{sel}(0, \text{cons}(X, L)) \rightarrow X \end{array} \right.$$

$$S \left\{ \begin{array}{l} \text{from}(N) \rightarrow \text{cons}(N, \text{from}(s(N))) \end{array} \right.$$

constructor
sharing

$$R_Z^\mu \left\{ \begin{array}{l} \text{sel}(s(N), \text{cons}(X, L)) \rightarrow \text{sel}(N, \text{activate}(L)) \\ \text{sel}(0, \text{cons}(X, L)) \rightarrow X \\ \text{activate}(X) \rightarrow X \end{array} \right.$$

non
composable!

$$S_Z^\mu \left\{ \begin{array}{l} \text{from}(N) \rightarrow \text{cons}(N, \text{from}'(s(N))) \\ \text{from}(N) \rightarrow \text{from}'(N) \\ \text{activate}(\text{from}'(N)) \rightarrow \text{from}(N) \\ \text{activate}(X) \rightarrow X \end{array} \right.$$



Modular termination of CSR



Disjoint unions

Modular termination of CSR

- **Definition:** Two CSRSs (R_1, μ_1) and (R_2, μ_2) are **compatible** if $\mu_1(f) = \mu_2(f)$ for shared symbols f . (Assumed from now on)
- **Definition:** A CSRS (R, μ) is **non-duplicating** if for every rule $l \rightarrow r$ and variable x in l , the **multiset of replacing occurrences** of x in r is included in the **multiset of replacing occurrences** of x in l

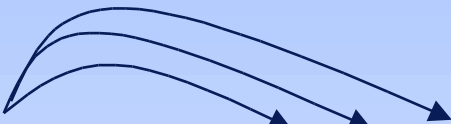
Modular termination of CSR

- **Theorem:** Termination of CSR is **modular** for **disjoint unions** of CSRSs (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) such that
 - ◆ both (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) are non-collapsing, or
 - ◆ both (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) are non-duplicating, or
 - ◆ either (\mathbf{R}_1, μ_1) or (\mathbf{R}_2, μ_2) is both non-collapsing and non-duplicating

Modular termination of CSR

■ Example:

duplicating


$$\mathbf{R}_1: f(a, b, x) \rightarrow f(x, x, x)$$
$$\mathbf{R}_2: g(x, y) \rightarrow x$$
$$g(x, y) \rightarrow y$$

collapsing

Modular termination of CSR

■ Example:

$$\mu_1(f) = \{3\}$$

$$\mu_2(g) = \{1, 2\}$$

$$\mathbf{R}_1: f(a, b, x) \rightarrow f(x, x, x)$$

$$\mathbf{R}_2: g(x, y) \rightarrow x$$
$$g(x, y) \rightarrow y$$

non-duplicating

non-duplicating

■ Now, $(\mathbf{R}_1, \mu_1) + (\mathbf{R}_2, \mu_2)$ is terminating!

Modular termination of CSR

- **Definition:** A CSRSs (\mathbf{R}, μ) is FP-terminating (or \mathbf{C} -terminating) if the CSRS

$$(\mathbf{R} + \{G(x, y) \rightarrow x, G(x, y) \rightarrow y\}, \mu),$$

where $\mu(G) = \{1, 2\}$ is terminating

- **Theorem:** FP-termination of CSR is **modular** for **disjoint unions** of CSRSs

Modular termination of CSR

- **Definition:** A CSRS (R, μ) is conservatively blocking (CB) if for every rule $l \rightarrow r$, whenever a variable x occurs at a non-replacing position in l , then x only occurs at non-replacing positions in r
- **Theorem:** For **disjoint unions** of CSRSs which **satisfy CB**, the following properties are modular

WN, WIN, and SIN

Weakly Normalizing

Weakly Innermost Normalizing

Innermost Terminating



Constructor-sharing unions

Modular termination of CSR

- **Definition:** Let (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) be constructor sharing CSRSs. A rule $l \rightarrow r$ is
 - ◆ **shared constructor lifting** if $\text{root}(r)$ is a shared constructor; \mathbf{R}_i is called **shared constructor lifting** if it has a shared constructor lifting rule
 - ◆ **shared symbol lifting** if $\text{root}(r)$ is a variable or a shared constructor; \mathbf{R}_i is called **shared symbol lifting** if it has a shared symbol lifting rule; \mathbf{R}_i is **layer preserving** if it is not shared symbol lifting

Modular termination of CSR

- **Theorem:** Termination of CSR is **modular** for constructor-sharing unions of CSRSs (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) such that
 - ◆ both (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) are layer-preserving, or
 - ◆ both (\mathbf{R}_1, μ_1) and (\mathbf{R}_2, μ_2) are non-duplicating, or
 - ◆ either (\mathbf{R}_1, μ_1) or (\mathbf{R}_2, μ_2) is both layer-preserving and non-duplicating

Modular termination of CSR

- **Example:** We can give a **modular proof** of termination of the constructor-sharing union

EXAMPLE + ID-NAT

This is because the (terminating) CSRS

$$\text{ID-NAT} \left\{ \begin{array}{l} 0===0 \rightarrow \text{true} \\ s(x)===s(y) \rightarrow x===y \\ x===y \rightarrow \text{false} \\ \mu(===)=\mu(s)= \end{array} \right.$$

is **layer preserving** and **non-duplicating** (see the proceedings for further details)

Modular termination of CSR

■ Example [GM99]:

$$\mu_1(f) = \{3\}$$

$$R_1: f(a, b, x) \rightarrow f(x, x, x)$$

non-duplicating

$$R_2: c \rightarrow a$$
$$c \rightarrow b$$

non-duplicating

- We are also able to give a **modular proof** of termination of $R_1 + R_2$ (left open in [GM99])

Modularity of simple μ -termination

- **Theorem:** FP-termination of CSR is **modular** for constructor-sharing unions of CSRSs where the **shared constructors** are **fully replacing**
- Without fully replacing shared constructors the theorem does **not** hold!

Modularity of simple μ -termination

■ Example: Consider the CSRSs

$$R \left\{ \begin{array}{l} \text{zeros} \rightarrow 0:\text{zeros} \\ \mu(\cdot) = \{1\} \end{array} \right.$$

FP-terminating

$$R' \left\{ \begin{array}{l} \text{length}([\]) \rightarrow 0 \\ \text{length}(x:y) \rightarrow s(\text{length}(y)) \\ \mu(\cdot) = \mu(\text{length}) = \mu(s) = \{1\} \end{array} \right.$$

FP-terminating

$$R + R' \left\{ \begin{array}{l} \text{zeros} \rightarrow 0:\text{zeros} \\ \text{length}([\]) \rightarrow 0 \\ \text{length}(x:y) \rightarrow s(\text{length}(y)) \\ \mu(\cdot) = \mu(\text{length}) = \mu(s) = \{1\} \end{array} \right.$$

non-terminating!

$\text{length}(\underline{\text{zeros}}) \rightarrow_{\mu} \underline{\text{length}(0:\text{zeros})} \rightarrow_{\mu} s(\text{length}(\underline{\text{zeros}})) \rightarrow_{\mu} \dots$

Modular termination of CSR

- **Theorem:** For **constructor-sharing unions** of CSRSs which **satisfy CB**, the following properties are modular
 - ◆ Weak normalization (WN)
 - ◆ Weak innermost normalization (WIN)
 - ◆ Innermost termination (SIN)



Conclusions and future work

Conclusions

- We have investigated the modularity of several termination properties of CSR, for disjoint and constructor-sharing unions
- In general, known results generalize to CSR but some additional conditions are needed
- As for the unrestricted rewriting, modularity often helps to provide easier proofs of termination of CSR

Future work

- Modularity in more general combinations of CSRSs
 - ◆ composable,
 - ◆ hierarchical
 - ◆ ...



Modular Termination of Context-Sensitive Rewriting

Salvador Lucas
Universidad Politécnica de Valencia
`slucas@dsic.upv.es`

joint work with

Bernhard Gramlich
Technische Universität Wien
`gramlich@logic.at`