Higher-Order (Non-)Modularity

Claus Appel & Vincent van Oostrom & Jakob Grue Simonsen

RTA 2010

Outline

Modularity

The Game

We want to prove properties of "large" rewriting systems by splitting them into "small", manageable pieces.

- Basic tool: Define the "large" system as the *union* of the "small" pieces.
- Call a property P *modular* if: P holds for the union of two systems iff P holds for each of the systems.

Modularity

The game

- We denote by $A \uplus B$ the disjoint union of sets A and B, and we denote by $\mathcal{T}_0 \oplus \mathcal{T}_1 = (\Sigma_0 \uplus \Sigma_1, R_0 \uplus R_1)$ the disjoint union of the rewrite systems $\mathcal{T}_i = (\Sigma_i, R_i)$ for $i \in \{0, 1\}$.
- A property P of a class $\mathcal{C}$ of rewrite systems is *modular* if $P(\mathcal{T}_0 \oplus \mathcal{T}_1) \Leftrightarrow P(\mathcal{T}_0) \;\&\; P(\mathcal{T}_1)$ for all $\mathcal{T}_0, \mathcal{T}_1 \in \mathcal{C}$

Warmup example

Termination is not modular (Toyama '87)

The TRS

$$R_0 = \left\{ \begin{array}{lcl} g(x,y) & \to & x \\ g(x,y) & \to & y \end{array} \right\}$$

is terminating (terms get strictly smaller).

The TRS

$$R_1 = \{f(a,b,x) \to f(x,x,x)\}$$

is also terminating (no new redexes can be created).

But . . .

Warmup example

Termination is not modular (Toyama '87)

$$R_0 = \left\{ \begin{array}{lcl} g(x,y) & \rightarrow & x \\ g(x,y) & \rightarrow & y \end{array} \right\}$$

$$R_1 = \{f(a,b,x) \rightarrow f(x,x,x)\}$$

In $R_0 \oplus R_1$:

$$\begin{array}{lcl} \underline{f(a,b,g(a,b))} & \rightarrow & f(\underline{g(a,b)}, g(a,b), g(a,b)) \\ & \rightarrow & f(a, \underline{g(a,b)}, g(a,b)) \\ & \rightarrow & f(a,b,g(a,b)) \end{array}$$

Modularity

Modularity has a long and varied history

Conditional rewriting, context-sensitive rewriting, graph rewriting, infinitary rewriting . . .

Mostly studied for first-order rewriting and its variants.

Modularity

But what about higher-order constructs?

Lambda calculus:

$$(\lambda x.M)\, N \rightarrow M\{N/x\}$$

or map:

$$\begin{array}{rcl} \mathrm{map}(F, \mathrm{nil}) & \rightarrow & \mathrm{nil} \\ \mathrm{map}(F, \mathrm{cons}(X, XS)) & \rightarrow & \mathrm{cons}(F(X), \mathrm{map}(F, XS)) \end{array}$$

Topic of today's talk.

Outline

Higher-Order Rewriting

Two possible extensions of first-order TRSs

- Variables can occur *applied* in terms: $X(a, b)$
- Terms can have *bound variables*: $\lambda x.x\, x$.

These are orthogonal to each other! (We can have one without the other without too much hassle.)

Higher-Order Rewriting

But in general, both extensions are used

$$(\lambda x.Z)\, W \rightarrow Z\{W/x\}$$

is often written as a *combinatory reduction system* (CRS):

$$\mathrm{app}(\mathrm{abs}([x]Z(x)), W) \rightarrow Z(W)$$

Higher-Order Rewriting

Same with map

As an STTRS:

$$\begin{array}{rcl} \mathrm{map}(F, \mathrm{nil}) & \to & \mathrm{nil} \\ \mathrm{map}(F, \mathrm{cons}(X, XS)) & \to & \mathrm{cons}(F(X), \mathrm{map}(F, XS)) \end{array}$$

As a CRS:

$$\begin{array}{rcl} \mathrm{map}(F, \mathrm{nil}) & \to & \mathrm{nil} \\ \mathrm{map}([x]F(x), \mathrm{cons}(X, XS)) & \to & \mathrm{cons}(F(X), \mathrm{map}(F, XS)) \end{array}$$

Various flavours of higher-order rewriting

No *generally* accepted single format

In this paper:

- *Combinatory Reduction Systems* (CRSs), Klop $\sim$ '80.
- *Pattern Rewrite Systems* (PRSs), Nipkow $\sim$ '90.
- *Simply Types TRSs* (STTRSs), Yamada $\sim$ '00 (no bound variables).

(+ applicative TRSs — not in this talk, though.)

Both CRSs and PRSs use *patterns* (consequence: Left-hand sides of rules have no nesting of meta-variables or appliation of meta-variables to function symbols). Thus, we can have $f([x]Z(x)) \to \mathrm{rhs}$, but not $X([x]Z(x)) \to \mathrm{rhs}$.

Commonalities

Features common to the standard higher-order formats

- Function symbols and variables are (simply) *typed* to constrain term formation (in particular, $X(X)$ is usually not allowed — instead use $\mathrm{app}(X, X)$).
- Every TRS is a higher-order system in any of the formats (good design!)
- If no bound variables $\Rightarrow$ examples can usually be translated from one format to the other.
- Bound variables $\Rightarrow$ examples from CRSs and PRSs can *usually* be translated to each other.

Outline

Nothing good ever lasts

The (short) story in (ordinary) first-order rewriting

Property	TRS
Confluence	Yes
Normalization	Yes
Termination	No
Completeness	No
Completeness, for left-linear systems	Yes
Unique normal forms	Yes

Nothing good ever lasts

Attack confluence and normalization!

Every counterexample for first-order systems is *also* a counterexample for higher-order systems. So: A non-modular property of TRSs is *also* non-modular i higher-order systems. *Confluence* and *Normalization* are modular for TRSs. However:

- *Neither* property is modular for *any* of the higher-order formats.

Confluence

Counterexample

$$R_0 = \{\mu\, Z \to Z\,(\mu\, Z)\}$$
$$R_1 = \{f\; W\; W \to a, f\; W\,(s\, W) \to b\}$$
But:
$$a \leftarrow f\,(\mu\, s)\,(\mu\, s) \to f\,(\mu\, s)\,(s\,(\mu\, s)) \to b.$$

Variations on an old theme: Klop essentially had the counterexample down in his 1980 PhD thesis.
Note: Example has no bound variables.

Confluence

Great! What if one of the systems has no rules (and application is shared)?

Confluence is not preserved under signature extension

$$R_0 = \left\{ \begin{array}{rcl} f(f(W)) & \to & f(W) \\ f([x]Z(x)) & \to & f(Z(a)) \\ f([x]Z(x)) & \to & f([x]Z(Z(x))) \end{array} \right\}$$

is confluent (use induction on terms)
But after extending the signature with a unary g:

$$f(g(a)) \leftarrow f([x]g(x)) \to f([x]g(g(x))) \to f(g(g(a)))$$

All is not lost: Left-linearity saves confluence

Theorem

Confluence is modular for left-linear systems.

Proof: Standard orthogonality argument using the Hindley-Rosen Lemma.
Not new: Known since the early 1990ies (see e.g. van Oostrom's 1994 PhD thesis, or earlier papers by Nipkow).

Normalization is not modular

Counterexample for mod. of norm. for PRSs

$$R_0 = \left\{ \begin{array}{lcl} f(x.Z(x), y.y) & \to & f(x.Z(x), y.Z(Z(y))) \\ f(x.x, y.Z(y)) & \to & a \end{array} \right\}$$

is normalizing (shown by induction on terms).

$$R_1 = \{g(g(x)) \to x\}$$

is *also* normalizing.

But ...

$$f(x.g(x), y.y) \leftrightarrow f(x.g(x), y.g(g(y)))$$

A plethora of counterexamples

In the paper: ∼15 counterexamples

Property	TRS	STTRS	CRS	PRS
Confluence	Yes	No	No	No
Normalization	Yes	No (†)	No (†)	No (†)
Termination	No	No	No	No
Completeness	No	No	No	No
Confluence, for left-linear systems	Yes	Yes	Yes	Yes
Completeness, for left-linear systems	Yes	No (†)	No (†)	No (†)
Unique normal forms	Yes	No (†)	No (†)	No (†)

Outline

The trouble with modularity proofs

Basic technique

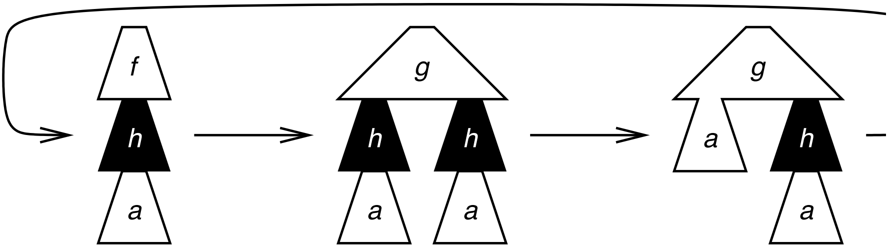
Decompose terms into maximal monochrome components—“chunks” of the term containing only symbols from *one* system.

(See picture on the next slide)

The trouble with modularity proofs

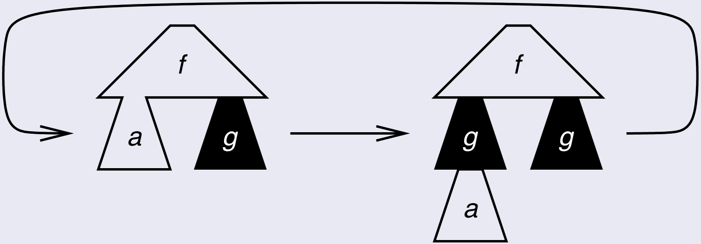
The key to most positive results in first-order systems

The *rank* of a term is the maximal number of signature changes in paths from the root to leaves.
The rank is non-decreasing across reduction . . .



The trouble with modularity proofs

Problem: *Exceedingly* hard to do for higher-order systems



The "white" system is $R_0 = \{f\,a\,Z \rightarrow f\,(Z\,a)\,Z\}$ and the "black" system is $R_1 = \{g\,W \rightarrow W\}$

Thee-pronged attack strategy

For higher-order systems:

Decompose-and-type

- Abstract away to consider reductions in an algebra of *components*.
- Include *type sizes* in the definition of rank.
- Use the *sum* instead of the *max* in the definition of rank.

(We can handle *variable application* this way, but not *bound variables*.)
Furter restriction: Simply Typed Term Rewriting Systems with *pattern* left-hand sides.

Components (no types yet!)

Components

For γ either black or white, a γ*-component* is a non-empty context built from γ-symbols and $\overline{\gamma}$-holes, which does not have active holes, *i.e. holes are not applied*.

Example

$f\,\blacksquare\,\blacksquare$ and $f\,(f\,\blacksquare)$ are 0-components, and b, $g\,g$ and $g\,(g\,(g\,\square))$ are 1-components.

Non-examples: $\blacksquare$ (empty), $f\,g$ (symbols of mixed colors), $\blacksquare\,\blacksquare$ (active hole), $f\,\square$ (same color symbol and hole), and $f\,\blacksquare\,\square$.

Components and Component-Type-Size

Abstract away

The set of components form a well-behaved algebra. Intuition: Instead of terms being made from the function symbols of R_0 and R_1, think of them as made from “symbols” that are really components.

The reflection of a term t as a “component term” is written in **bold** as $\mathbf{t}$ in the following.

Think: Terms are trees of black and white legos.

Components and Component-Type-Size

First-order equivalent

"Rank":

$$\begin{aligned} \#\boldsymbol{t} &= 0 && \text{if there is only one component} \\ \#\boldsymbol{C}(\vec{\boldsymbol{t}}) &= 1 + \max_i(\#\boldsymbol{t}_i) \end{aligned}$$

(Observe: No typing, no summation, just *max*.

Components and Component-Type-Size

Component-type size

The component-type size, $|t|$, of term t is defined to be the pair $|\boldsymbol{t}|$ defined by:

$$|\boldsymbol{C}(\vec{\boldsymbol{t}})| = (\gamma \cdot \#\tau + \#\vec{\boldsymbol{t}}, \overline{\gamma} \cdot \#\tau + \#\vec{\boldsymbol{t}}) \quad \text{if } C : \tau \text{ has color } \gamma$$

where

$$\begin{aligned} \#b &= 1 \\ \#(\sigma \to \tau) &= \#\sigma + 1 + \#\tau \\ \#\boldsymbol{C}(\vec{\boldsymbol{t}}) &= \#\tau + \#\vec{\boldsymbol{t}} \qquad \text{if} C : \tau \end{aligned}$$

(So $\#$ looks very much like the size of simple types!). *Very important*: The *max* from first-order rewriting has been replaced by a *sum* (implicit in $\#\vec{\boldsymbol{t}}$) .

Lo and behold!

Non-duplicatingness

A rewrite rule is *non-duplicating* if no free (meta-)variable occurs more often in its right-hand side than in its left-hand side.

Lemma

If $t \to s$ in the disjoint union of non-duplicating pattern STTRSs, then $|t| \geq |s|$.

(Proof by tedious induction in an auxiliary lemma, *critically* employing non-duplicatingness.)
Also, $|t| > |s|$ when two or more components are *amalgamated* in a step.

Consequence

So: The component-type size in pattern STTRSs works "just like" rank in first-order TRSs.

Consequence

Key Lemma(s)

The following hold:

- The rewrite relation $\rightarrow$ induces a rewrite relation $\Rightarrow$ on *component terms*.
- If $t \rightarrow s$ and $|t| = |s|$ in the disjoint union of non-duplicating pattern STTRSs, then $\boldsymbol{t} \Rightarrow \boldsymbol{s}$.

Consequences

Consequence I

Termination is modular for non-duplicating pattern STTRSs.

Proof: Choose, for contradiction, an infinite reduction in $R_0 \oplus R_1$ starting from a term of *minimal* component-type size.
$\Rightarrow$ terminates on terms if R_0 and R_1 terminate(!)
Simulate $\rightarrow$ by $\Rightarrow$ using key lemma from previous slide.

Consequences

Consequence II

Normalization is modular for non-duplicating pattern STTRSs.

Proof: As before.

Consequences

What about bound variables?

No go!

Termination is *not* modular for non-duplicating PRSs. The presence of bound variables can “simulate” dupliation (due to substitution), even if the system is non-duplicating — see example in the paper.

Outline

New bits in the paper: (†)

Whole paper in one table

Property	TRS	STTRS	CRS	PRS
Confluence	Yes	No	No	No
Normalization	Yes	No (†)	No (†)	No (†)
Termination	No	No	No	No
Completeness	No	No	No	No
Confluence, for left-linear systems	Yes	Yes	Yes	Yes
Completeness, for left-linear systems	Yes	No (†)	No (†)	No (†)
Unique normal forms	Yes	No (†)	No (†)	No (†)
Norm., non-dup. pat. systems	Yes	Yes (†)	?	?
Term., non-dup. pat. systems	Yes	Yes (†)	?	No (†)

Future work (all of you, and all of us)

- Cheap: Fill the holes in the previous table!
- Find *other* sufficient conditions for modularity of any interesting property.
- Find some way to encompass bound variables. Obvious possibility: Use a linear substitution calculus or only consider the subset of linear terms.
- Retrofitting: Use the method of component algebras to re-derive classic modularity results.
- Find special restricted systems: Very few interesting higher-order systems are right-linear (we want to be able to handle map, fold, etc.)

The final item above is probably the most important.

?