

Size-Based Termination: Semantics and Generalizations

Cody Roux

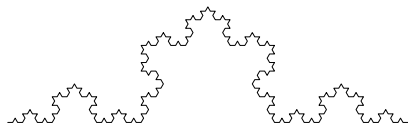
INRIA-Lorraine Pareo

June 7, 2011

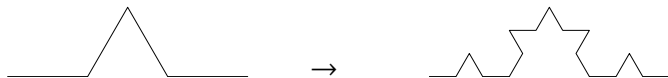
Outline

Introduction

High-level computation is about **recognizing shapes**:

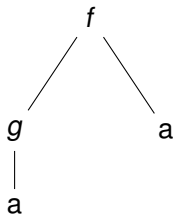


And **modifying** them:



We are particularly interested in **tree structures** and their transformations.

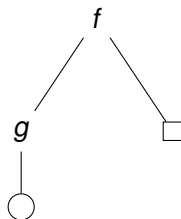
Trees can be represented using **algebraic data-types**:
the **tree**



Is represented by the **term**:

$$f(g(a), a)$$

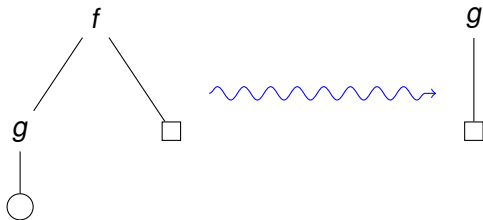
We can also represent **families** of trees using terms with **variables**:
The family of trees:



is represented by the term

$$f(g(x), y)$$

We can then represent **transformations** using pairs of terms:
 The transformation:



Is represented by the **rule**:

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

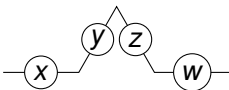
Applying the transformation is called **rewriting**.

Example: the snowflakes can be represented by terms of the shape

- *Line*



- $Peak(x, y, z, w)$



For example if $a = Peak(Line, Line, Line, Line)$ then

$Peak(a, Line, Line, a)$ represents: 

We can transform such a shape by the rule:

$$\textit{Line} \rightarrow \textit{Peak}(\textit{Line}, \textit{Line}, \textit{Line}, \textit{Line})$$

This is **non-deterministic**: applied to $\textit{Peak}(\textit{Line}, \textit{Line}, \textit{Line}, \textit{Line})$ it can give



or



Rewriting can therefore encode **decisions**.

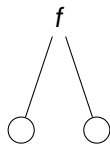
More formally:

- A **term** is either a **variable** x, y, \dots or a **function symbol** f, g, \dots applied to a number of terms, depending on its **arity**.
- A **rewrite rule** is a pair $l \rightarrow r$ of terms such that all variables of r are present in l .
- A **substitution** θ is a (partial) mapping from variables to terms. We write $t\theta$ for the term t in which every variable x is replaced with $\theta(x)$.
- A term t **rewrites** to a term u if there is a rule $l \rightarrow r$, a subterm t' and a substitution θ such that $t' = l\theta$ and u is equal to t with the occurrence of t' replaced with $r\theta$.

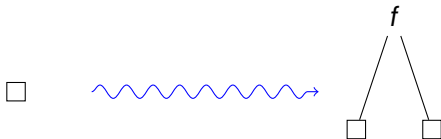
A few remarks:

- Pure rewriting has a very rich and profound theory [Bezem et al., 2003, Jouannaud, 1993, Dershowitz and Jouannaud, 1990, Baader and Nipkow, 1998].
- Rewriting is very **expressive**: it is Turing-complete, and most problems are undecidable: Termination, Confluence, Reachability... And can serve as a model for computation, security protocols, etc.
- Rewriting can be seen as a semantics for **equality**: a rewrite rule can be seen as a **directed** equation.

Given a family of terms, for example:



There is a **fundamental** transformation:



This is the **substitution** operation.

We wish to **reify** this transformation.

This transformation becomes the **term**

$$\lambda x.f(x, x)$$

Called the **abstraction** of $f(x, x)$. This term can be **applied**:

$$(\lambda x.f(x, x))t$$

rewrites to

$$f(t, t)$$

This reduction is called β -reduction.

The combination of β -reduction and rewriting is very expressive: it may form the basis of a real programming language!
(e.g. Ocaml/SML [Cousineau et al., 1985, Milner et al., 1997], Haskell [Hudak et al., 1992], Elan [Borovanský et al., 1998], Maude [Clavel et al., 1996]...)
Our approach is founded on [Jouannaud and Okada, 1991].

In fact a bit **too** expressive!

We add types:

- Allows simple algebraic semantics
- Makes β -reduction alone **terminating**

How can we use this additional structure to give guarantees?



Outline

Algebraic semantics

Size-based termination

Semantic Labelling

The Model Construction

Proving Termination

We adopt the algebraic semantics for higher-order systems described by Plotkin, Fiore and Turi [Fiore et al., 1999] and extended to rewriting by Hamana [Hamana, 2003].



What is an algebraic structure?

The categorical view:

An algebraic structure is a functor which encodes information which can refer to its argument:

$$\text{Nat}(X) = \text{Unit} + X$$

An element of $\text{Nat}(X)$ is either a “special element” i or a box containing an $e \in X$.

Given a functor F , an F -algebra A is an interpretation of $F(A)$ into A :

$$\text{eval}: F(A) \rightarrow A$$



For $\text{Nat}(X)$ over **sets**:

An algebra is a set A with

- An element $z \in A$, the image of i by $eval$
- A function $s : A \rightarrow A$ which sends the boxed element e to $s(e)$.

- Nat -algebras form a **category**
- The **initial** Nat -algebra is the set of **closed terms**

$$0, S(0), S(S(0)), \dots$$

Or, equivalently, \mathbb{N} .

How do we encode information about **variable binding**?

Idea: replace sets with **variable indexed** sets, called **presheafs**:

For each set $\mathcal{V} = \{x_1, \dots, x_n\}$ of variables $A(\mathcal{V})$ are the elements of A that **depend** on x_1, \dots, x_n .

Motivating example:

$$\mathit{Term}(\mathcal{V}) = \{t \mid \text{free variables of } t \subseteq \mathcal{V}\}$$

Abstraction then becomes a morphism of presheafs:

$$abs_{\mathcal{V}}: A(\mathcal{V} \cup \{x\}) \rightarrow A(\mathcal{V})$$

for $x \notin \mathcal{V}$.

We define the presheaf

$$A^x(\mathcal{V}) = A(\mathcal{V} \cup \{x\})$$

for $x \notin \mathcal{V}$.

This gives

$$abs: A^x \rightarrow A$$

In the typed framework:

$$\frac{}{\Gamma, x: T, \Delta \vdash x: T} \text{ var}$$

$$\frac{}{\Gamma \vdash f: \tau_f} \text{ sig}$$

$$\frac{\Gamma \vdash t: T \rightarrow U \quad \Gamma \vdash u: T}{\Gamma \vdash t u: U} \text{ app}$$

$$\frac{\Gamma, x: T \vdash t: U}{\Gamma \vdash \lambda x: T. t: T \rightarrow U} \text{ abs}$$



The typed algebras: presheafs over **contexts** instead of sets of variables and indexed by **types**.

The term presheaf

$$\mathit{Term}_T(\Gamma) = \{t \mid \Gamma \vdash t : T\}$$

We can give the signature for the that defines **term algebra** functor:

$$\Sigma_{\lambda}(F) = V + S + F \times F + F^{\times}:-$$

We can give the signature for the that defines **term algebra** functor:

$$\Sigma_\lambda(F) = \underbrace{V}_{\text{var}} + \underbrace{S}_{\text{sig}} + \underbrace{F \times F}_{\text{app}} + \underbrace{F^x: -}_{\text{abs}}$$

We can give the signature for the that defines **term algebra** functor:

$$\Sigma_{\lambda}(F) = \underbrace{V}_{\text{var}} + \underbrace{S}_{\text{sig}} + \underbrace{F \times F}_{\text{app}} + \underbrace{F^{\times} \text{ :-}}_{\text{abs}}$$

Note: the actual product and abstraction are slightly more complex.

We define the operator \bullet on presheafs:

$$(A \bullet B)_T(\Gamma)$$

is the set of tuples

$$(a, b_1, \dots, b_n)$$

such that

$$a \in A_T(x_1 : T_1, \dots, x_n : T_n) \Rightarrow b_i \in B_{T_i}(\Gamma)$$

modulo renaming.

This can be seen as the *unapplied* substitution

$$a\{x_1 \mapsto b_1, \dots, x_n \mapsto b_n\}$$



The operator \bullet defines a **monoidal product** for term algebras.
We define a **substitution** to be an arrow

$$\text{subst} : A \bullet A \rightarrow A$$

which turns A into a \bullet -monoid.

We furthermore require the structure to be **compatible** with the algebra structure; we call this a **\bullet -algebra**.

Theorem (Fiore, Plotkin & Turi)

Term is the **initial** \bullet -algebra.

This gives us an **interpretation morphism**

$$(\llbracket _ \rrbracket): \mathit{Term} \rightarrow A$$

for every \bullet -algebra A .

We write:

$$(\llbracket \Gamma \vdash t : A \rrbracket) \in A_T(\Gamma)$$

Finally, we want this structure to be compatible with the rewrite rules: A is an **ordered** presheaf such that

$$t \rightarrow_{\mathcal{R} \cup \beta}^* u \quad \Rightarrow \quad (t) \geq (u)$$

We call such a structure a **premodel**, or **model** if there is **equality**.

Question

How can we use models to simplify termination problems?

Outline

Algebraic semantics

Size-based termination

Semantic Labelling

The Model Construction

Proving Termination



Size-based termination uses **type annotations** to deduce **size-information** about terms.

We have a judgment for base types:

$$\Gamma \vdash_{\text{size}} t : B^a$$

Which denotes: t is of **size** a .



We consider **strictly positive** inductive types: if c is a constructor of the type B , for example

$$c : A \rightarrow B \rightarrow (A \rightarrow B) \rightarrow B$$

then we annotate with size information to give

$$c : A \rightarrow B^\alpha \rightarrow (A \rightarrow B^\beta) \rightarrow B^{\max(\alpha, \beta) + 1}$$

The size of a term intuitively corresponds to the size of the **normal form** as a **tree**.



The size-type system is given by the following rules
 [Blanqui and Roux, 2009, Blanqui, 2004, Barthe et al., 2004]:

$$\frac{}{\Gamma, x : T, \Delta \vdash_{\text{size}} x : T} \text{ax}$$

$$\frac{\phi \text{ subst}}{\Gamma \vdash_{\text{size}} f : \tau_f \phi} \text{symb}$$

$$\frac{\Gamma \vdash_{\text{size}} t : T \rightarrow U \quad \Gamma \vdash_{\text{size}} u : T}{\Gamma \vdash_{\text{size}} t u : U} \text{app}$$

$$\frac{\Gamma, x : T^\infty \vdash_{\text{size}} t : U}{\Gamma \vdash_{\text{size}} \lambda x : |T|. t : T \rightarrow U} \text{abs}$$

$$\frac{\Gamma \vdash_{\text{size}} t : T \quad T \leq U}{\Gamma \vdash_{\text{size}} t : U} \text{sub}$$



The criterion then requires (among other things) that each rule

$$f l_1 \dots l_n \rightarrow r$$

if $\Gamma \vdash_{\text{size}} l_i : B^{a_i}$ then

$$\Gamma \vdash_{\text{size}}^{\vec{a}} r : T$$

where $\vdash_{\text{size}}^{\vec{a}}$ constrains **recursive calls** on f to **smaller arguments**.

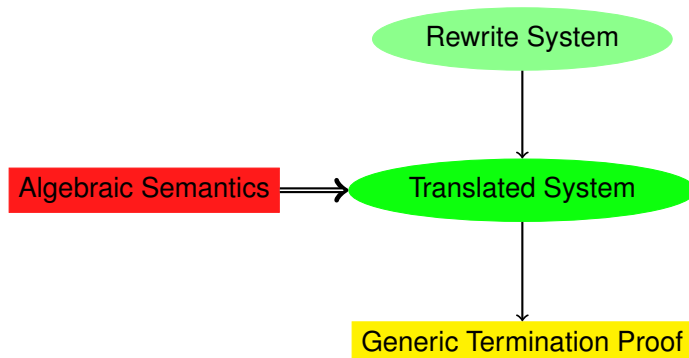


The traditional proof of correctness involves a modification of the classic Tait normalization proof.

We desire

- A more **conceptually simple** proof.
- A more **modular** proof.

Our approach:





Outline

Algebraic semantics

Size-based termination

Semantic Labelling

Currying

Stability by Reduction

The Model Construction

Proving Termination



Our transformation method needs to use **semantic information** in the terms.

We select **semantic labelling** [Zantema, 1995].



The idea:

- Select a **model** $\mathcal{M}_{\mathcal{R}}$ of the rewrite system \mathcal{R} , with interpretation $(_)_{\mathcal{M}}$.
- In each rule $l \rightarrow r \in \mathcal{R}$, we replace each $f(t_1, \dots, t_n) \in l, r$ with

$$f_{(\!(t_1)\!), \dots, (\!(t_n)\!)}(t_1, \dots, t_n)$$

recursively.

- Show that **normalization** of the new system is **equivalent** to that of the original system.



Hamana [Hamana, 2007] describes such a framework for the **higher-order** case and shows correctness.



However his approach has several drawbacks:

- Application may not be **curried**.
- The reduction associated to β -reduction is **not** β -reduction!



The definition of labelling:

Without currying

ϕ is a valuation and

- $\overline{x}^\phi = x$
- $\overline{t\ u}^\phi = \overline{t}^\phi\ \overline{u}^\phi$
- $\overline{\lambda x: T.t}^\phi = \lambda x: T.\overline{t}^{\phi_x}$
- $\overline{f(t_1, \dots, t_n)}^\phi = f_{(\overline{t})_\phi}(\overline{t_1}^\phi, \dots, \overline{t_n}^\phi)$

Where ϕ_x^x **weakens** the context for ϕ and **extends** it to send x to x .
 The labelling of a term only depends on the semantics of **subtypes**.



In the curried framework:

$f(t_1, t_2, \dots, t_n)$ is represented by $(\dots ((f t_1) t_2) \dots t_n)$

The labelling of this last term is

$$f_{()} \overline{t_1}^\phi \dots \overline{t_n}^\phi$$

There is no meaningful label for f as it takes no arguments!



Our solution: **non structural labelling**:

$$\overline{f t_1 \dots t_n}^\phi = f_{(\vec{t})_\phi} \overline{t_1}^\phi \dots \overline{t_n}^\phi$$

f may be applied to **less** than n arguments.

$$\overline{f t_1 \dots t_k}^\phi = f_{((t_1)_\phi, \dots, (t_k)_\phi, (?))} \overline{t}^\phi$$



We solve this by weakening the context for f -labels in this case:

$$\overline{f t_1 \dots t_k}^\phi = f((t_1)_{\phi'}, \dots, (t_k)_{\phi'}, (x_1)_{\phi'}, \dots, (x_{n-k})_{\phi'}) \overline{t}^{\overline{\phi}}$$

with $\phi' = \phi_{\substack{x_1 \dots x_{n-k} \\ x_1 \dots x_{n-k}}}$



Now the fundamental property of labelling in the first-order case is expressed by:

$$t \rightarrow_{\mathcal{R}} t' \Leftrightarrow \forall \phi, \bar{t}^{\phi} \rightarrow_{\bar{\mathcal{R}}} \bar{t}'^{\phi}$$

if we are working with a model and

$$t \rightarrow_{\mathcal{R}} t' \Leftrightarrow \forall \phi, \bar{t}^{\phi} \rightarrow_{\bar{\mathcal{R}} \cup \text{Decr}^*} \bar{t}'^{\phi}$$

if we are working with a premodel, with

$$\text{Decr} = \{f_l \rightarrow f_{l'} \mid l \geq_{\mathcal{M}^*} l'\}$$



In the higher-order framework we would like to have

$$t \rightarrow_{\mathcal{R} \cup \beta} t' \Leftrightarrow \bar{t}^\phi \rightarrow_{\bar{\mathcal{R}} \cup \beta} \bar{t}'^\phi$$

to be able to apply some generic termination argument to the labelled system.

However this property **fails** in Hamana's framework.



It fails in 2 different ways:

instantiation of variables in a context

$$(\lambda x : T. f x) t \rightarrow_{\beta} f t$$

labelling gives

$$(\lambda x : T. f_{(x \mapsto x)} x) \bar{t} \not\rightarrow_{\beta} f_{(t)} \bar{t}$$

And symmetrically:

substitution of a term into a context

$$(\lambda y : T. \lambda x : U. y) (f t) \rightarrow_{\beta} \lambda x : U. f t$$

labelling gives

$$(\lambda y : T. \lambda x : U. y) (f_{(t)} t) \not\rightarrow_{\beta} \lambda x : U. f_{(x \mapsto t)} t$$



To solve this failure we introduce two orders on labels:

$$(|x \vdash t|) >_{inst} (|t|)_{x \mapsto v}$$

Which allows **instantiation** of free variables in labels and

$$(|t|) >_{weak} (|x \vdash t|)$$

which allows us to **weaken** the context of the labels.

And allow decrease of the labels in rewriting:

$$\text{Struct} = \{f_l \rightarrow f_{l'} \mid l >_{inst,decr} l'\}$$



With the structural rules, we can “save” the result:

Theorem:

$$t \rightarrow_{\mathcal{R} \cup \beta} t' \quad \Leftrightarrow \quad \bar{t}^\phi \rightarrow_{\bar{\mathcal{R}} \cup \beta \cup \text{Decr}^* \cup \text{Struct}^*} \bar{t}'^\phi$$



However we **lose termination**, as Struct is non-terminating: If x is free in t , then

$$(x \vdash t) >_{inst} (t) >_{weak} (x \vdash t)$$

We can however express a **relative termination result**:

Corollary:

$$t \text{ is } \mathcal{R} \cup \beta\text{-normalizing} \iff \bar{t}^\phi \text{ is } \overline{\mathcal{R}}\beta\text{-normalizing} \\ \text{relative to } \text{Decr} \cup \text{Struct}$$

Outline

Algebraic semantics

Size-based termination

Semantic Labelling

The Model Construction

Proving Termination

We want to build a **set-theoretical** model in which:

- Abstractions are interpreted by **functions**.
- There is a natural notion of **size**.

We proceed by **cumulativity**.

We mutually define:

- elements of inductive types by **tuples**

$$(\mathbf{c} \ t_1 \ \dots \ t_n)_\phi = (\mathbf{c}, (t_1)_\phi, \dots, (t_n)_\phi)$$

- elements of arrow types by **functions**

$$(\lambda x : T. t)_\phi = v \mapsto (t)_{\phi_v^x}$$

We need to find **interpretations** for base types.

Pure set theoretic extensions lead to very large sets!

Solution

Use **realizable** function spaces.

We consider the interpretation:

$$\llbracket A \rightarrow B \rrbracket = \{f \in \llbracket B \rrbracket^{\llbracket A \rrbracket} \mid \exists t, t \Vdash f\}$$

with

$$t \Vdash f \Leftrightarrow \forall u \Vdash x, t \ u \Vdash f(x)$$



Using this definition and the **Tarski fixed-point theorem** we can interpret types.

There is a natural notion of **size**:

- $\text{size}((c, t_1, \dots, t_n)) = \max(\text{size}(t_1), \dots, \text{size}(t_n)) + 1$
- $\text{size}(f) = \sup_x f(x)$



The size types correspond to sizes in the model:

$$\Gamma \vdash_{\text{size}} t : B^a \quad \Rightarrow \quad \theta \models \mu \Rightarrow \text{size}(\llbracket t \rrbracket_\theta) \leq \llbracket a \rrbracket_\mu$$

By well-founded induction on the sizes we can interpret **defined functions**

$$f_{Alg}(x_1) \dots (x_n) = \llbracket r \rrbracket_\theta$$

for $f\vec{l} \rightarrow r \in \mathcal{R}$ and $\llbracket \vec{l} \rrbracket_\theta = \vec{x}$. We use the **orthogonality** restriction here.



Using this construction we build a **model** Alg of the rewrite system.

$$Alg_T(\Gamma) = \llbracket \Gamma \rightarrow T \rrbracket = \llbracket T_1 \rrbracket \rightarrow \dots \rightarrow \llbracket T_n \rrbracket \rightarrow \llbracket T \rrbracket$$

$$subst(f, x_1, \dots, x_n) = v \mapsto f(x_1(v)) \dots (x_n(v))$$

etc.

Theorem:

Alg constitutes a $\mathcal{R} \cup \beta$ -model.



Outline

Algebraic semantics

Size-based termination

Semantic Labelling

The Model Construction

Proving Termination



For the termination proof, we need a **relative precedence termination** criterion.

We proceed in a similar manner to [Blanqui, 2003].



We consider a typed left-algebraic higher-order rewrite system \mathcal{R} and a typed algebraic rewrite system \mathcal{S} . We suppose that

- The positivity conditions are satisfied.
- There is a well-founded precedence $>_{prec}$ on the function symbols.
- The rules in \mathcal{R} **respect** the precedence:

$$\forall f \vec{l} \rightarrow r \in \mathcal{R}, g \in r \Rightarrow f >_{prec} g$$

- The precedence is **compatible** with \mathcal{S} .

Compatibility states: if

$$f t_1 \dots t_n \rightarrow_S g u_1 \dots u_m$$

then

- \vec{u} is an **S-permutation** of \vec{t} :

$$\forall i \exists j, t_j \rightarrow_S u_i$$

- g is **weaker** for $>_{prec}$ than f :

$$\forall h, g >_{prec} h \Rightarrow f >_{prec} h$$

Theorem:

If \mathcal{R} and \mathcal{S} satisfy the criterion then

$$\Gamma \vdash t : T \quad \Rightarrow \quad t \in SN$$



Define $>_{decr}$ on labels to be

$$f_l >_{decr} f_{l'} \iff l >_{Alg} l'$$

with

$$x >_{Alg} y \iff \text{size}(x) > \text{size}(y)$$

To show termination of the size-based system we take

$$f >_{prec} g \iff f >_{Struct} \circ >_{decr} g$$

It is easy to show compatibility.

Well-foundedness is more difficult: we may have

$$\begin{array}{ccccc} \Gamma_1 & \supseteq & \Gamma_2 & \subseteq & \Gamma_3 \\ x_1 & >_{prec} & x_2 & >_{prec} & x_3 \end{array}$$

In particular **lexicographic** ordering on sizes and the size of contexts is **insufficient**.

We use a lemma from Doornbos and Von Karger [Doornbos and Karger, 1998].

This method is **generic**: it can be used to prove different termination results, for example:



Theorem (Breazu-Tannen, Gallier & Okada)[Breazu-Tannen and Gallier, 1990, Okada, 1989]:

Let \mathcal{R} be a **first-order** (uni-sorted) rewrite system **that is SN**.

Then the system consisting of

- A single base type D .
- **Curried** typed rewrite rules: if f is of arity n then

$$f: \underbrace{D \rightarrow \dots \rightarrow D}_n \rightarrow D$$

and

$$f(l_1, \dots, l_n) \rightarrow r \quad \mapsto \quad f \text{ curry}(l_1) \dots \text{curry}(l_n) \rightarrow \text{curry}(r)$$

- β -reduction

Is **strongly normalizing** on typed terms.

However this approach **fails** on certain simple rewrite systems.

There is **no model** for which

$$f (S x) \rightarrow (\lambda y: T.f y) x$$

can be shown to be terminating in the labelling framework.

We need to be capable of analyzing **control flow**.

To do this we **go back to the types**. We need a system capable of analyzing **potential calls**.

In first-order rewriting, this is achieved using **dependency pairs**.

We wish to capture part of the analysis on the **dependency graph** using **refinement types**

We restrict ourselves to the type of **unlabeled binary trees**.

The idea:

The **refinement** $\mathbf{B}(p)$ of the type \mathbf{B} of trees is

$$\mathbf{B}(p) = \{t \in \mathbf{B} \mid t \text{ is of shape } p\}$$

We perform analysis on the **shapes**.

Outline

The Type System

Dependency Graph

The Termination Criterion

The Termination Semantics

Perspectives

We give the following shapes or **patterns**:

- The top pattern \top , which denotes any possible tree.
- The **leaf** pattern which denotes leaves.
- The **node**(p, q) pattern which denotes trees which are nodes with left subtree of shape p and right subtree of shape q .
- The bottom pattern \perp which denotes no possible tree.
- Variables α which allow us to quantify over all patterns.

We can then describe our system.

types:

$$T, U \in \mathcal{T} := \mathbf{B}(p) \mid \forall \alpha. T \mid T \rightarrow U$$

terms:

$$t, u \in \mathcal{T}rm := x \mid f \mid \lambda x: T. t \mid \lambda \alpha. t \mid t u \mid t p \mid \text{Leaf} \mid \text{Node}$$

Note that abstraction and application of patterns is **explicit**.

Contrary to the previous approach, we only treat matching on **non-defined terms**, but with no orthogonality restriction.

We define a type system to assign types with patterns to terms.

$$\begin{array}{c}
 \frac{}{\Gamma, x: T, \Gamma' \vdash x: T} \mathbf{ax} \\
 \frac{\Gamma, x: T \vdash t: U}{\Gamma \vdash \lambda x: T. t: T \rightarrow U} \mathbf{t-lam} \\
 \frac{}{\Gamma \vdash \mathbf{Leaf}: \mathbf{B}(\mathbf{leaf})} \mathbf{leaf-intro} \\
 \frac{}{\Gamma \vdash \mathbf{Node}: \forall \alpha \beta. \mathbf{B}(\alpha) \rightarrow \mathbf{B}(\beta) \rightarrow \mathbf{B}(\mathbf{node}(\alpha, \beta))} \mathbf{node-intro} \\
 \frac{\Gamma \vdash t: T}{\Gamma \vdash \lambda \alpha. t: \forall \alpha. T} \mathbf{p-lam}
 \end{array}$$

with α free in Γ .

$$\begin{array}{c}
 \frac{\Gamma \vdash t: T \rightarrow U \quad \Gamma \vdash u: T}{\Gamma \vdash t u: U} \mathbf{t-app} \\
 \frac{\Gamma \vdash t: \forall \alpha. T}{\Gamma \vdash t p: T\{\alpha \mapsto p\}} \mathbf{p-app} \\
 \frac{}{\Gamma \vdash f: \tau_f} \mathbf{symb}
 \end{array}$$

This is insufficient in general to type interesting rewrite systems:

$$\begin{aligned} \text{rev Leaf} &\rightarrow \text{Leaf} \\ \text{rev (Node } x \ y) &\rightarrow \text{Node (rev } y) \ (\text{rev } x) \end{aligned}$$

The type of *rev* can only be $\forall \alpha. \mathbf{B}(\alpha) \rightarrow \mathbf{B}(\top)$ and we need **subtyping** to derive

$$\text{Leaf} : \mathbf{B}(\top)$$

and

$$\text{Node (rev } y) \ (\text{rev } x) : \mathbf{B}(\top)$$

We introduce **subtyping rules**

We define the sub-pattern relation \ll by:

- $p \ll p$
- $p \ll \top$
- $\perp \ll p$
- $p_1 \ll q_1 \wedge p_2 \ll q_2 \Rightarrow \text{node}(p_1, p_2) \ll \text{node}(q_1, q_2)$

This leads to the following subtyping:

- $p \ll q \Rightarrow \mathbf{B}(p) \leq \mathbf{B}(q)$
- $T_2 \leq T_1 \wedge U_1 \leq U_2 \Rightarrow T_1 \rightarrow U_1 \leq T_2 \rightarrow U_2$
- $T \leq U \Rightarrow \forall \alpha. T \leq \forall \alpha. U$

$$\frac{\Gamma \vdash t : T \quad T \leq U}{\Gamma \vdash t : U} \text{sub}$$

Thanks to **explicit annotations** of pattern abstraction and application, all of our rules are **syntax directed**, except for the **subtyping rules**.

An alternate application rule:

$$\frac{\Gamma \vdash t : T \rightarrow U \quad \Gamma \vdash u : T' \quad T' \leq T}{\Gamma \vdash t u : U} \text{sub - app}$$

This rule replaces **application** and **subtyping**.

Using **transitivity** of subtyping we show that the new rules are equivalent to the old ones.

Theorem:

Type inference is **decidable**.

Outline

The Type System

Dependency Graph

The Termination Criterion

The Termination Semantics

Perspectives

We suppose that every rule can be **well-typed** in the framework and that:

- Each symbol f has a number of **recursive arguments** of type \mathbf{B} .
- The type of f is of the form

$$f: \forall \alpha_1 \dots \alpha_n. \mathbf{B}(\alpha_1) \rightarrow \dots \rightarrow \mathbf{B}(\alpha_n) \rightarrow T_f$$

- the α_j appear **positively** in T_f .
- In each rule $l \rightarrow r$, each symbol $g \in r$ is **fully applied** to its type arguments.

We wish to analyze the types involved in typing to build a **type based dependency graph**.

The dependency pairs:

For each rule $f \vec{\alpha} \vec{l} \rightarrow r$ and each $g \in r$ such that

- $\Gamma \vdash_{min} l_i : \mathbf{B}(p_i)$
- $g \ q_1 \dots q_n$ appears in r

We build the dependency pair

$$f^\#(p_1, \dots, p_n) \rightarrow g^\#(q_1, \dots, q_m)$$

We need to check possible **successive calls** by examination of the pairs. To do this we define **pattern unification** \bowtie .

- A variable α pattern-unifies with **everything**.
- \top pattern-unifies with **everything**.
- \perp may unify with a **variable**, with \top or **itself**
- Similarly **leaf** unifies with a variable, \top or **leaf**.
- Same for **node**(p, q) which unifies with a variable, \top , or **node**(p', q') iff

$$p \bowtie p' \quad \wedge \quad q \bowtie q'$$

We write:

$$f^\#(p_1, \dots, p_n) \bowtie f^\#(q_1, \dots, q_n)$$

if

$$p_1 \bowtie q_1 \wedge \dots \wedge p_n \bowtie q_n$$

The **dependency graph** has

- as **nodes** the **dependency pairs**
- an **edge** between $l^\# \rightarrow r^\#$ and $l'^\# \rightarrow r'^\#$ if

$$r^\# \bowtie l'^\#$$

We can also define a **decrease** \triangleright on patterns:

The **closure by context** of

node $p, q \triangleright p$

and

node $p, q \triangleright q$

We carry this to dependency pairs:

$$f(\text{node}(p, q)) \rightarrow^{\triangleright} g(p)$$

Outline

The Type System

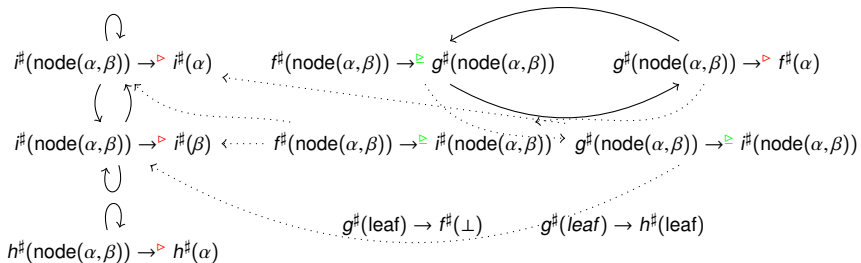
Dependency Graph

The Termination Criterion

The Termination Semantics

Perspectives

To show termination we observe the dependency graph:



We need to verify that each cycle contains only **weak decreases** (marked with \triangleright) and at least one **strict decrease** (marked with \triangleright).

Theorem [Roux, 2011]:

If the above condition is satisfied, every **well-typed term** is **strongly normalizing**

Outline

The Type System

Dependency Graph

The Termination Criterion

The Termination Semantics

Perspectives

To prove termination, we modify the classic proof by **candidates**: we need to find an interpretation of the base types.

In particular we need to **instantiated** pattern variables. We instantiate them with **closed patterns**.

The intuition:

$$\llbracket \mathbf{B}(p) \rrbracket_{\theta} = \{t \mid t \text{ matches } p\}$$

Problem

If a non-confluent term

Node (Node x y) $z \leftarrow t \rightarrow$ Node Leaf Leaf

is in $\mathbf{B}(\text{node}(\alpha, \beta))$, then we would necessarily have $\alpha \mapsto \top$.

This is not sufficiently **precise**.

We interpret pattern variables by **sets of closed patterns**.

If t is a term in \mathcal{SN} and P is a set of **closed patterns**, we write $t \downarrow \ll P$ if

For each **normal form** u of t , there is **some** $p \in P$ such that u matches p .

We take θ to send variables to **sets** of closed patterns. We define

$$\llbracket \mathbf{B}(p) \rrbracket_{\theta} = \{t \in \mathcal{SN} \mid t \downarrow \ll p\theta\}$$

Then we carry out the classic construction

- $\llbracket A \rightarrow B \rrbracket_\theta = \{t \mid \forall u \in \llbracket A \rrbracket_\theta, t \ u \in \llbracket B \rrbracket_\theta\}$
- $\llbracket \forall \alpha. A \rrbracket_\theta = \{t \mid \forall P, t \in \llbracket T \rrbracket_{\theta_P^\alpha}\}$

It **works**.

To show **decrease in terms** we need additional information.

Lemma:

If $t = (\text{Node } l_1 \ l_2)\sigma$, then every normal form of t is of the shape

Node $v_1 \ v_2$

with v_1 and v_2 normal forms of $l_1\sigma$ and $l_2\sigma$.

From this we get

- if $t = (\text{Node } l_1 \ l_2)\sigma$
- if for every θ

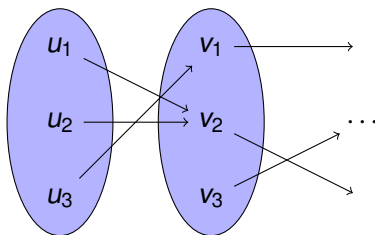
$$t \in \llbracket \mathbf{B}(\text{node}(\alpha, \beta)) \rrbracket_{\theta} \quad \Rightarrow \quad u \in \llbracket \mathbf{B}(\alpha) \rrbracket_{\theta}$$

Then every normal form of u is smaller than some normal form of t .

Infinite sequences of calls:

$$t_1 \rightarrow t_2 \rightarrow \dots$$

therefore give rise to sequences of normal forms



And we may conclude by **König's lemma**.

Outline

The Type System

Dependency Graph

The Termination Criterion

The Termination Semantics

Perspectives

We have described two **distinct** extensions to the **size-types** approach to termination of higher-order rewrite systems. We would like a **combination** which

- Has the power of the **algebraic semantics**.
- Can capture the notions of **control flow**.

The Objective

Prove **completeness** of such a framework.

In addition, for both these frameworks, we can go **up**:

- More expressive type theories.
- Weaker conditions: relax **orthogonality**, matching on **defined constructors**, **higher-order** inductive types.

A natural extension of the type-based dependency framework is to allow **unions** of base types:

$$\mathbf{B}(p_1) \cup \dots \cup \mathbf{B}(p_n)$$

and explore the possible **lattices of types** that can be authorized.






We can also look at **conversion** at the type level:

$$f: \forall \alpha. \mathbf{B}(\alpha) \rightarrow \mathbf{B}(\tilde{f}(\alpha)), \quad \tilde{f}(\text{leaf}) \simeq p, \quad \tilde{f}(\text{node}(\gamma, \delta)) \simeq q$$

and study the type annotations as a **first order rewrite system**.

Look for **type-level** analogues of **first-order** techniques (interpretations, simplification orderings).

Thank you!

-  Baader, F. and Nipkow, T. (1998).
Term Rewriting and All That.
Cambridge University Press.
-  Barthe, G., Frade, M. J., Giménez, E., Pinto, L. and Uustalu, T. (2004).
Type-based termination of recursive definitions.
Mathematical Structures in Computer Science 14, 97–141.
-  Bezem, M., Klop, J. W. and de Vrijer, R., eds (2003).
Term Rewriting Systems.
Cambridge Tracts in Theoretical Computer Science,
Cambridge University Press.
-  Blanqui, F. (2003).
Rewriting Modulo in Deduction Modulo.
In RTA, (Nieuwenhuis, R., ed.), vol. 2706, of Lecture Notes in
Computer Science pp. 395–409, Springer.
-  Blanqui, F. (2004).

A type-based termination criterion for dependently-typed higher-order rewrite systems.

In *Proceedings of RTA'04*, (van Oostrom, V., ed.), vol. 3091, pp. 24–39,.



Blanqui, F. and Roux, C. (2009).

On the Relation between Sized-Types Based Termination and Semantic Labelling.

In *CSL*, (Grädel, E. and Kahle, R., eds), vol. 5771, of *Lecture Notes in Computer Science* pp. 147–162, Springer.



Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.-E. and Ringeissen, C. (1998).

An Overview of ELAN.

In *Proc. of WRLA* vol. 15, *Electronic Notes in Theoretical Computer Science*.



Breazu-Tannen, V. and Gallier, J. (1990).

Polymorphic rewriting conserves algebraic strong normalisation.

Theoretical Computer Science 83, 3–28.



Clavel, M., Eker, S., Lincoln, P. and Meseguer, J. (1996).
Principles of Maude.

In Proc. of WRLA vol. 4, Electronic Notes in Theoretical
Computer Science.



Cousineau, G., Curien, P.-L. and Mauny, M. (1985).
The Categorical Abstract Machine.

In FPCA pp. 50–64,.



Dershowitz, N. and Jouannaud, J.-P. (1990).
Rewrite Systems.

In Handbook of Theoretical Computer Science, (Leeuwen, J. v.,
ed.), vol. B: Formal Models and Semantics, chapter 6, pp.
243–320.



Doornbos, H. and Karger, B. V. (1998).
On the Union of Well-Founded Relations.



Fiore, M. P., Plotkin, G. D. and Turi, D. (1999).

Abstract Syntax and Variable Binding.

In *LICS* pp. 193–202,.



Hamana, M. (2003).

Term rewriting with variable binding: an initial algebra approach.

In *PPDP* pp. 148–159, ACM.



Hamana, M. (2007).

Higher-Order Semantic Labelling for Inductive Datatype Systems.

In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*.



Hudak, P., Peyton Jones, S., Wadler, P., Arvind, B., Fairbairn, B. J., Fasel, J., Guzmán, M., Hammond, K., Hughes, J., Johnsson, T., Kieburtz, R., Nikhil, R., Partain, W. and Peterson, J. (1992).

Report on the functional programming language Haskell,
version 1.2.

Special Issue of SIGPLAN Notices 27.



Jouannaud, J.-P. (1993).

Introduction to Rewriting.

In Term rewriting, (Comon, H. and Jouannaud, J.-P., eds), vol.
909, pp. 1–15,.



Jouannaud, J.-P. and Okada, M. (1991).

Executable higher-order algebraic specification languages.

In Proceedings of LICS'91 pp. 350–361, IEEE Computer
Society Press.



Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997).

The Definition of Standard ML (Revised).

The MIT Press.



Okada, M. (1989).

Strong normalizability for the combined system of the typed
 λ -calculus and an arbitrary convergent term rewrite system.

In Proc. of ISSAC pp. 357–363, ACM Press.



Roux, C. (2011).

Refinement Types as Higher-Order Dependency Pairs.

In 22nd International Conference on Rewriting Techniques and Applications (RTA'11), (Schmidt-Schauß, M., ed.), vol. 10, of Leibniz International Proceedings in Informatics (LIPIcs) pp. 299–312, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.



Zantema, H. (1995).

Termination of Term Rewriting by Semantic Labelling.

Fundamenta Informaticae 24, 89–105.