Size-Based Termination: Semantics and Generalizations

Cody Roux

INRIA-Lorraine Pareo

June 7, 2011
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:

```
    f
   / \  
  g   |
     / 
    o
```

is represented by the term

\[ f(g(x), y) \]
We can then represent **transformations** using pairs of terms:

The transformation:

\[ f(g(x), y) \rightarrow g(y) \]

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 \(a = \text{Peak}(\text{Line}, \text{Line}, \text{Line}, \text{Line})\) then

\(\text{Peak}(a, \text{Line}, \text{Line}, a)\) represents:
We can transform such a shape by the rule:

\[ \text{Line} \rightarrow \text{Peak} (\text{Line}, \text{Line}, \text{Line}, \text{Line}) \]

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

or

Rewriting can therefore encode decisions.
More formally:

- A term is either a variable $x, y, \ldots$ or a function symbol $f, g, \ldots$ 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 $\theta$ 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 $\theta$ such that $t' = l\theta$ and $u$ is equal to $t$ with the occurrence of $t'$ replaced with $r\theta$. 
A few remarks:

- 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:

\[ f \]

There is a fundamental transformation:

\[ f \]

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 \( \beta \)-reduction.
The combination of $\beta$-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 $\beta$-reduction alone terminating

How can we use this additional structure to give guarantees?
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic semantics</td>
</tr>
<tr>
<td>Size-based termination</td>
</tr>
<tr>
<td>Semantic Labelling</td>
</tr>
<tr>
<td>The Model Construction</td>
</tr>
<tr>
<td>Proving Termination</td>
</tr>
</tbody>
</table>
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:

\[ Nat(X) = Unit + X \]

An element of \( Nat(X) \) 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 \( Nat(X) \) over \textbf{sets}:
An algebra is a set \( A \) with

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

- \( Nat \)-algebras form a \textbf{category}
- The \textbf{initial} \( Nat \)-algebra is the set of \textbf{closed terms}

\[
0, S(0), S(S(0)), \ldots
\]

Or, equivalently, \( \mathbb{N} \).

How do we encode information about \textbf{variable binding}?
Idea: replace sets with variable indexed sets, called presheafs:

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

Motivating example:

$$\text{Term}(\mathcal{V}) = \{t \mid \text{free variables of } t \subseteq \mathcal{V}\}$$
Abstraction then becomes a morphism of presheafs:

\[ \text{abs}_V : A(V \cup \{x\}) \to A(V) \]

for \( x \notin V \).

We define the presheaf

\[ A^x(V) = A(V \cup \{x\}) \]

for \( x \notin V \).

This gives

\[ \text{abs} : A^x \to A \]
In the typed framework:

\[
\begin{align*}
&\Gamma, x : T, \Delta \vdash x : T \quad \text{var} \\
&\Gamma \vdash f : \tau_f \quad \text{sig} \\
&\Gamma \vdash t : T \rightarrow U \quad \Gamma \vdash u : T \quad \Gamma \vdash t \, u : U \quad \text{app} \\
&\Gamma, x : T \vdash t : U \quad \Gamma \vdash \lambda x : T. t : T \rightarrow U \quad \text{abs}
\end{align*}
\]
The typed algebras: presheafs over contexts instead of sets of variables and indexed by types.

The term presheaf

\[ \text{Term}_T(\Gamma) = \{ t \mid \Gamma \vdash t : T \} \]
We can give the signature for the that defines term algebra functor:

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

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

Note: the actual product and abstraction are slightly more complex.
We define the operator \( \bullet \) on presheaves:

\[
(A \bullet B)_\Gamma
\]

is the set of tuples

\[
(a, b_1, \ldots, b_n)
\]

such that

\[
a \in A_{\Gamma}(x_1 : T_1, \ldots, x_n : T_n) \Rightarrow b_i \in B_{T_i}(\Gamma)
\]

modulo renaming.

This can be seen as the *unapplied* substitution

\[
a \{x_1 \mapsto b_1, \ldots, x_n \mapsto b_n\}
\]
The operator $\bullet$ defines a monoidal product for term algebras. We define a substitution to be an arrow

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

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

\[
(\_): \text{Term} \rightarrow A
\]

for every \( \bullet \)-algebra \( A \).

We write:

\[
(\Gamma \vdash t : A) \in A_T(\Gamma)
\]
Finally, we want this structure to be compatible with the rewrite rules: \( A \) is an ordered presheaf such that

\[
t \rightarrow^*_{R \cup \beta} u \implies (|t|) \geq (|u|)
\]

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

How can we use models to simplify termination problems?
<table>
<thead>
<tr>
<th>Algebraic semantics</th>
<th>Size-based termination</th>
<th>Semantic Labelling</th>
<th>The Model Construction</th>
<th>Proving Termination</th>
</tr>
</thead>
</table>

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
[Blanqué and Roux, 2009, Blanqué, 2004, Barthe et al., 2004]:

\[
\begin{align*}
\Gamma, x : T, \Delta & \vdash \text{size } x : T \\
\phi \text{ subst} & \\
\Gamma & \vdash \text{size } f : \tau_f \phi \\
\Gamma & \vdash \text{size } t : T \rightarrow U \quad \Gamma & \vdash \text{size } u : T \\
\Gamma & \vdash \text{size } t \; u : U \\
\Gamma, x : T^\infty & \vdash \text{size } t : U \\
\Gamma & \vdash \text{size } \lambda x : |T|. t : T \rightarrow U \\
\Gamma & \vdash \text{size } t : T \quad T \leq U \\
\Gamma & \vdash \text{size } t : U
\end{align*}
\]
The criterion then requires (among other things) that each rule

\[ f \ l_1 \ldots l_n \to r \]

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

\[ \Gamma \vdash \bar{a} : r : T \]

where \( \vdash_{\text{size}} \) constrains \textit{recursive calls} on \( f \) to \textit{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 $f(t_1, \ldots, t_n) \in l, r$ with
  
  $$f((t_1), \ldots, (t_n))(t_1, \ldots, t_n)$$

  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 $\beta$-reduction is not $\beta$-reduction!
The definition of labelling:

**Without currying**

$\phi$ is a valuation and

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

Where $\phi^x_x$ weakens the context for $\phi$ and extends it to send $x$ to $x$. The labelling of a term only depends on the semantics of subtypes.
In the curried framework:

\[ f(t_1, t_2, \ldots, t_n) \] is represented by \((\ldots((f \ t_1) \ t_2) \ldots t_n)\)

The labelling of this last term is

\[ f() \ t_1^\phi \ldots t_n^\phi \]

There is no meaningful label for \(f\) as it takes no arguments!
Our solution: non structural labelling:

\[
\phi \overline{f \, t_1 \ldots t_n} = \phi \overline{f(t_1)} \overline{t_1} \ldots \overline{t_n}
\]

\(f\) may be applied to less than \(n\) arguments.

\[
\phi \overline{f \, t_1 \ldots t_k} = \phi \overline{f(t_1)} \overline{\ldots} \overline{f(t_k)} \overline{\ldots} \overline{t}
\]
We solve this by weakening the context for $f$-labels in this case:

$$f \ t_1 \ldots t_k \phi = f((t_1)_{\phi'}, \ldots, (t_k)_{\phi'}, (x_1)_{\phi'}, \ldots, (x_{n-k})_{\phi'})$$

with $\phi' = \phi x_1 \ldots x_{n-k}$
Now the fundamental property of labelling in the first-order case is expressed by:

\[ t \rightarrow_{R} t' \Leftrightarrow \forall \phi, \overline{t}^\phi \rightarrow_{\overline{R}} \overline{t'}^\phi \]

if we are working with a model and

\[ t \rightarrow_{R} t' \Leftrightarrow \forall \phi, \overline{t}^\phi \rightarrow_{\overline{R} \cup \text{Decr}^*} \overline{t'}^\phi \]

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

\[ t \rightarrow_{R \cup \beta} t' \iff t^\phi \rightarrow_{R \cup \beta} t'^\phi \]

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

\[(\lambda x : T. f_{(x \mapsto x)} \; x) \; t \not\rightarrow_{\beta} \; f_{(t)} \; t\]

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:

\[(| x \vdash t |) >_{\text{inst}} (| t |)_{x \mapsto v}\]

Which allows instantiation of free variables in labels and

\[(| t |) >_{\text{weak}} (| x \vdash t |)\]

which allows us to weaken the context of the labels.

And allow decrease of the labels in rewriting:

\[\text{Struct} = \{ f_l \rightarrow f_{l'} \mid l >_{\text{inst,decr}} l' \}\]
With the structural rules, we can “save” the result:

**Theorem:**

\[ t \xrightarrow{\mathcal{R} \cup \beta} t' \iff \overline{t}^\phi \xrightarrow{\mathcal{R} \cup \beta \cup \text{Decr}^* \cup \text{Struct}^*} \overline{t'}^\phi \]
However we lose termination, as Struct is non-terminating: If $x$ is free in $t$, then

$$\vdash x \vdash t \vdash t \vdash t \vdash t$$

We can however express a relative termination result:

**Corollary:**

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

  \[
  (\langle c \ t_1 \ldots \ t_n \rangle_\phi = (c, (t_1)_\phi, \ldots, (t_n)_\phi)
  \]

- elements of arrow types by **functions**

  \[
  (\lambda x : T.t)_\phi = \nu \mapsto (t)_{\phi^x}\nu
  \]

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:

\[ [A \rightarrow B] = \{ f \in [B]^{[A]} | \exists t, t \vdash f \} \]

with

\[ t \vdash f \iff \forall u \vdash x, t u \vdash f(x) \]
Using this definition and the **Tarski fixed-point theorem** we can interpret types.

There is a natural notion of **size**:  
- $size((c, t_1, \ldots, t_n)) = \max(size(t_1), \ldots, size(t_n)) + 1$  
- $size(f) = \sup_x f(x)$
The size types correspond to sizes in the model:

$$\Gamma \vdash \text{size } t : B^a \quad \Rightarrow \quad \theta \models \mu \Rightarrow \text{size}(\langle t \rangle_\theta) \leq \langle a \rangle_\mu$$

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

$$f_{\text{Alg}}(x_1) \ldots (x_n) = \langle r \rangle_\theta$$

for $$f\overline{l} \rightarrow r \in \mathcal{R}$$ and $$\langle \overline{l} \rangle_\theta = \overline{x}$$. We use the orthogonality restriction here.
Using this construction we build a model $Alg$ of the rewrite system.

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

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

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 $\mathcal{S}$. The Criterion We suppose that

- The positivity conditions are satisfied.
- There is a well-founded precedence $>_{\text{prec}}$ on the function symbols.
- The rules in $\mathcal{R}$ respect the precedence:
  \[ \forall f \vec{l} \rightarrow r \in \mathcal{R}, g \in r \implies f >_{\text{prec}} g \]
- The precedence is compatible with $\mathcal{S}$. 
Compatibility states: if

\[ f \ t_1 \ldots t_n \rightarrow_S g \ u_1 \ldots u_m \]

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 \( S \) satisfy the criterion then

\[ \Gamma \vdash t : T \quad \Rightarrow \quad t \in SN \]
Define $\succ_{decr}$ on labels to be

$$f_l \succ_{decr} f_{l'} \iff l \succ_{Alg} l'$$

with

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

To show termination of the size-based system we take

$$f \succ_{prec} g \iff f \succ_{Struct \circ} \succ_{decr} g$$

It is easy to show compatibility.
Well-foundedness is more difficult: we may have

\[ \Gamma_1 \supseteq \Gamma_2 \subseteq \Gamma_3 \]
\[ x_1 >_{prec} x_2 >_{prec} x_3 \]

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
  \[
  f : D \rightarrow \ldots \rightarrow D \rightarrow D
  \]
  and
  \[
  f(l_1, \ldots, l_n) \rightarrow r \quad \leftrightarrow \quad f \text{ curry}(l_1) \ldots \text{curry}(l_n) \rightarrow \text{curry}(r)
  \]
- $\beta$-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 $B(p)$ of the type $B$ of trees is

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

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 $\bot$ which denotes no possible tree.
- Variables $\alpha$ which allow us to quantify over all patterns.
We can then describe our system.

**types:**

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

**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.

\[
\Gamma, x : T, \Gamma' \vdash x : T \\
\Gamma, x : T \vdash t : U \\
\Gamma \vdash \lambda x : T.t : T \rightarrow U \\
\Gamma \vdash \text{Leaf} : B(\text{leaf}) \\
\Gamma \vdash \text{Node} : \forall \alpha \beta. B(\alpha) \rightarrow B(\beta) \rightarrow B(\text{node}(\alpha, \beta)) \\
\Gamma \vdash t : T \\
\Gamma \vdash \lambda \alpha. t : \forall \alpha. T \\
\Gamma \vdash t : T \rightarrow U \quad \Gamma \vdash u : T \\
\Gamma \vdash t\ u : U \\
\Gamma \vdash t : \forall \alpha. T \\
\Gamma \vdash t\ p : T\{\alpha \mapsto p\} \\
\Gamma \vdash f : \tau_f
\]

with \(\alpha\) free in \(\Gamma\).
This is insufficient in general to type interesting rewrite systems:

\[
\begin{align*}
\text{rev } \text{Leaf} & \rightarrow \text{Leaf} \\
\text{rev } (\text{Node } x \ y) & \rightarrow \text{Node } (\text{rev } y) \ (\text{rev } x)
\end{align*}
\]

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

\[
\text{Leaf} : B(\top)
\]

and

\[
\text{Node } (\text{rev } y) \ (\text{rev } x) : B(\top)
\]
We introduce **subtyping rules**

We define the sub-pattern relation \( \ll \) by:

- \( p \ll p \)
- \( p \ll \top \)
- \( \bot \ll p \)
- \( p_1 \ll q_1 \land p_2 \ll q_2 \Rightarrow \text{node}(p_1, p_2) \ll \text{node}(q_1, q_2) \)

This leads to the following subtyping:

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

\[
\Gamma \vdash t : T \quad T \leq U \\
\hline
\Gamma \vdash t : U \quad \text{sub}
\]
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:

\[
\frac{\Gamma \vdash t : T \rightarrow U \quad \Gamma \vdash u : T' \quad T' \leq T}{\Gamma \vdash t \ u : U} \quad \text{sub – app}
\]

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.*
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 $B$.
- The type of $f$ is of the form
  \[ f : \forall \alpha_1 \ldots \alpha_n. B(\alpha_1) \to \ldots \to B(\alpha_n) \to T_f \]
- the $\alpha_i$ appear positively in $T_f$.
- In each rule $l \to r$, 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 $f \bar{\alpha} \xrightarrow{\bar{l}} r$ and each $g \in r$ such that
- $\Gamma \vdash_{min} l_i : B(p_i)$
- $g q_1 \ldots q_n$ appears in $r$

We build the dependency pair

$$f^#(p_1, \ldots, p_n) \rightarrow g^#(q_1, \ldots, q_m)$$
We need to check possible successor calls by examination of the pairs. To do this we define pattern unification ▷.

- A variable $\alpha$ pattern-unifies with everything.
- $\top$ pattern-unifies with everything.
- $\bot$ 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

$$p ▷ p' \land q ▷ q'$$

We write:

$$f^\#(p_1, \ldots, p_n) ▷ f^\#(q_1, \ldots, q_n)$$

if

$$p_1 ▷ q_1 \land \ldots \land p_n ▷ q_n$$
The dependency graph has
- as nodes the dependency pairs
- an edge between $l^# \rightarrow r^#$ and $l'^# \rightarrow r'^#$ if

\[
 r^# \gg l'^# 
\]
We can also define a decrease $\triangleright$ on patterns:

The closure by context of

$$\text{node } p, q \triangleright p$$

and

$$\text{node } p, q \triangleright q$$

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:

\[
[B(p)]_\theta = \{ t \mid t \text{matches } p \}
\]

**Problem**

If a non-confluent term

\[
\text{Node (Node } x \ y) \ z \leftarrow t \rightarrow \text{Node Leaf Leaf}
\]

is in \(B(\text{node}(\alpha, \beta))\), 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 $\theta$ to send variables to sets of closed patterns. We define

$$\llbracket B(p) \rrbracket_\theta = \{ t \in \mathcal{SN} \mid t \Downarrow \ll p \theta \}$$
Then we carry out the classic construction

- \([A \rightarrow B]_\theta = \{ t \mid \forall u \in [A]_\theta, t \ u \in [B]_\theta \}\)
- \([\forall \alpha. A]_\theta = \{ t \mid \forall P, t \in [T]_{\theta_P}^\alpha \}\)

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 \( \text{Node } v_1 v_2 \)

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 \( \theta \)

\[
\begin{align*}
t \in \left[ B(\text{node}(\alpha, \beta)) \right]_\theta & \quad \Rightarrow \quad u \in \left[ B(\alpha) \right]_\theta
\end{align*}
\]

Then every normal form of \( u \) is smaller than some normal form of \( t \).
Infinite sequences of calls:

\[ t_1 \rightarrow t_2 \rightarrow \ldots \]

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:

$$B(p_1) \cup \ldots \cup B(p_n)$$

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

We can also look at conversion at the type level:

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

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!
Term Rewriting and All That.
Cambridge University Press.

Type-based termination of recursive definitions.
Mathematical Structures in Computer Science 14, 97–141.

Term Rewriting Systems.

Rewriting Modulo in Deduction Modulo.

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

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.

An Overview of ELAN.

Polymorphic rewriting conserves algebraic strong normalisation.


In FPCA pp. 50–64.


Abstract Syntax and Variable Binding.
In LICS pp. 193–202,

Term rewriting with variable binding: an initial algebra approach.
In PPDP pp. 148–159, ACM.

Higher-Order Semantic Labelling for Inductive Datatype Systems.


Special Issue of SIGPLAN Notices 27.

Introduction to Rewriting.

Executable higher-order algebraic specification languages.

The Definition of Standard ML (Revised).
The MIT Press.

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