Termination of term graph rewriting

Hans Zantema

Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen
Joined work with Dennis Nolte and Barbara König

Workshop on Termination, Obergurgl, September, 2016

Term rewriting

Term rewriting

A *term rewrite system* (TRS) is a set of rules $\ell \rightarrow r$, in which ℓ, r are *terms*, typically containing variables

Term rewriting

A *term rewrite system* (TRS) is a set of rules $\ell \to r$, in which ℓ, r are *terms*, typically containing variables

If t is a term, and σ maps variables to terms, then $t\sigma$ is the term obtained from t by replacing every variable x by $\sigma(x)$

Term rewriting

A *term rewrite system* (TRS) is a set of rules $\ell \to r$, in which ℓ, r are *terms*, typically containing variables

If t is a term, and σ maps variables to terms, then $t\sigma$ is the term obtained from t by replacing every variable x by $\sigma(x)$

Term rewriting = if a subterm is of the shape $\ell\sigma$ for some rule $\ell \to r$, then it may be replaced by $r\sigma$

Term rewriting

A *term rewrite system* (TRS) is a set of rules $\ell \to r$, in which ℓ, r are *terms*, typically containing variables

If t is a term, and σ maps variables to terms, then $t\sigma$ is the term obtained from t by replacing every variable x by $\sigma(x)$

Term rewriting $=$ if a subterm is of the shape $\ell\sigma$ for some rule $\ell \to r$, then it may be replaced by $r\sigma$

This is a widely applied standard way of computation

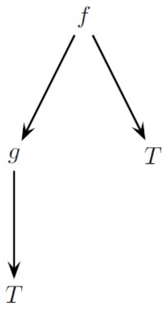
Sharing

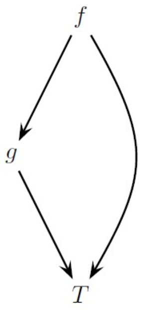
Sharing

For efficient implementation of terms and term rewriting a subterm occurring twice should be stored only once

Sharing

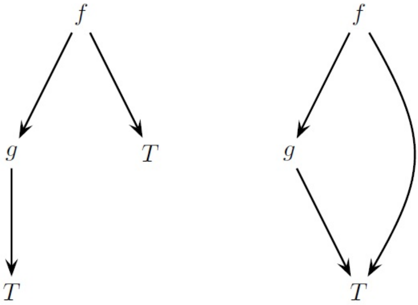
For efficient implementation of terms and term rewriting a subterm occurring twice should be stored only once





Sharing

For efficient implementation of terms and term rewriting a subterm occurring twice should be stored only once



In this way terms are represented by *DAGs* (directed acyclic graphs) rather than by trees

In rewriting DAGs it is natural not to duplicate dags

In rewriting DAGs it is natural not to duplicate dags

For instance, by

$$x * (y + z) \to (x * y) + (x * z)$$

in rewriting terms = trees, the subterm corresponding to x is duplicated, while in rewriting DAGs the subterm corresponding to x only gets an extra incoming arrow

In rewriting DAGs it is natural not to duplicate dags

For instance, by

$$x * (y + z) \to (x * y) + (x * z)$$

in rewriting terms = trees, the subterm corresponding to x is duplicated, while in rewriting DAGs the subterm corresponding to x only gets an extra incoming arrow

With the three rules

$$f(x, a, b) \to f(x, x, x), \; c \to a, \; c \to b$$

the infinite term reduction

$$f(c, a, b) \to f(c, c, c) \to f(c, a, c) \to f(c, a, b) \to \cdots$$

can not be mimicked in DAG rewriting without doing intermediate unsharing

So if variables are duplicated, not every term rewriting reduction can be mimicked by DAG rewriting.

So if variables are duplicated, not every term rewriting reduction can be mimicked by DAG rewriting.

If either the rewrite systems is non-duplicating, or it is orthogonal (no overlap between left hand sides), then every term rewriting reduction can be mimicked by DAG rewriting

So if variables are duplicated, not every term rewriting reduction can be mimicked by DAG rewriting.

If either the rewrite systems is non-duplicating, or it is orthogonal (no overlap between left hand sides), then every term rewriting reduction can be mimicked by DAG rewriting

In all cases we assume the rewrite system is *left-linear* (no duplicate variables in left hand sides), otherwise it is not clear how to rewrite

So if variables are duplicated, not every term rewriting reduction can be mimicked by DAG rewriting.

If either the rewrite systems is non-duplicating, or it is orthogonal (no overlap between left hand sides), then every term rewriting reduction can be mimicked by DAG rewriting

In all cases we assume the rewrite system is *left-linear* (no duplicate variables in left hand sides), otherwise it is not clear how to rewrite

What happens if we allow graphs with cycles?

So if variables are duplicated, not every term rewriting reduction can be mimicked by DAG rewriting.

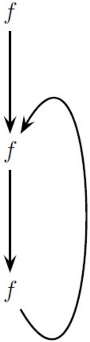
If either the rewrite systems is non-duplicating, or it is orthogonal (no overlap between left hand sides), then every term rewriting reduction can be mimicked by DAG rewriting

In all cases we assume the rewrite system is *left-linear* (no duplicate variables in left hand sides), otherwise it is not clear how to rewrite

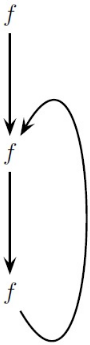
What happens if we allow graphs with cycles?

They may represent *infinite* terms

Example



Example



represents the infinite term

$$f^{\omega} = f(f(f(f(f(\cdots)))))$$

Term graphs

Such a graph is called a *term graph*, in which all nodes are labeled by operation symbols, and a node labeled by f of arity n has exactly n numbered outgoing edges

Such a graph is called a *term graph*, in which all nodes are labeled by operation symbols, and a node labeled by f of arity n has exactly n numbered outgoing edges

More precisely:

Definition

A term graph over a signature Σ is a triple $(V, \mathrm{lab}, \mathrm{succ})$ in which

- V is a finite set of nodes (vertices)
- $\mathrm{lab} : V \to \Sigma$ is a partial labeling function
- $\mathrm{succ} : V \to V^*$ is the partial successor function having the same domain as lab, such that $\forall v \in V : |\mathrm{succ}(v)| = \mathrm{ar}(\mathrm{lab}(v))$

Term graphs

Such a graph is called a *term graph*, in which all nodes are labeled by operation symbols, and a node labeled by f of arity n has exactly n numbered outgoing edges

More precisely:

Definition

A term graph over a signature Σ is a triple $(V, \mathrm{lab}, \mathrm{succ})$ in which

- V is a finite set of nodes (vertices)
- $\mathrm{lab} : V \to \Sigma$ is a partial labeling function
- $\mathrm{succ} : V \to V^*$ is the partial successor function having the same domain as lab, such that $\forall v \in V : |\mathrm{succ}(v)| = \mathrm{ar}(\mathrm{lab}(v))$

Terms are interpreted as term graphs where lab and succ are undefined for variables

This numbering is essential since we want to distinguish the term graphs corresponding to the terms $f(a,b)$ and $f(b,a)$

Term graph rewriting

Term graph rewriting

How to apply a term rewrite rule $\ell \to r$ on a term graph?

Term graph rewriting

How to apply a term rewrite rule $\ell \to r$ on a term graph?

Basic idea:

- Find a match of ℓ in the graph
- Add r to the graph, connecting root and variables to those of the match of ℓ

Term graph rewriting

How to apply a term rewrite rule $\ell \to r$ on a term graph?

Basic idea:

- Find a match of ℓ in the graph
- Add r to the graph, connecting root and variables to those of the match of ℓ

Precise definition is given by a *double push-out*:

$$\begin{tikzcd}
L \arrow[d, "g"'] & I \arrow[l, "\ell"'] \arrow[r, "r"] \arrow[d, "i"] & R \arrow[d, "h"] \\
G & D \arrow[l, "dg"'] \arrow[r, "dh"] & H
\end{tikzcd}$$

where L, R are the term graphs of ℓ, r, I is the *interface* (describing which parts of L are connected to which parts of R), and G is the graph that is rewritten to H

What to do with the rest of ℓ?

What to do with the rest of ℓ?

Two options:

What to do with the rest of ℓ?

Two options:

- Remove it: the *basic version*, or

What to do with the rest of ℓ?

Two options:

- Remove it: the *basic version*, or
- Keep it: the *extended version*

The extended version was studied before, and is a natural semantics if one is interested in the unfolded, possibly infinite term, and everything is considered modulo sharing/unsharing

What to do with the rest of ℓ?

Two options:

- Remove it: the *basic version*, or
- Keep it: the *extended version*

The extended version was studied before, and is a natural semantics if one is interested in the unfolded, possibly infinite term, and everything is considered modulo sharing/unsharing

However, when considering term graphs as finite objects, and implicit sharing or unsharing is not allowed, then the basic version is the most natural one

What to do with the rest of ℓ?

Two options:

- Remove it: the *basic version*, or
- Keep it: the *extended version*

The extended version was studied before, and is a natural semantics if one is interested in the unfolded, possibly infinite term, and everything is considered modulo sharing/unsharing

However, when considering term graphs as finite objects, and implicit sharing or unsharing is not allowed, then the basic version is the most natural one

cycle rewriting $\approx$ term graph rewriting in basic version with unary symbols only

Example

Is the single rule $f(g(x)) \to f(x)$ terminating?

Example

Is the single rule $f(g(x)) \to f(x)$ terminating?

In the basic version: *yes*, since in every step a g-node is removed

Example

Is the single rule $f(g(x)) \rightarrow f(x)$ terminating?

In the basic version: *yes*, since in every step a g-node is removed

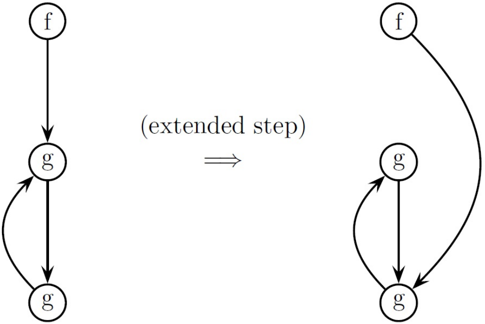
In the extended version: *no*

Example

Is the single rule $f(g(x)) \to f(x)$ terminating?

In the basic version: *yes*, since in every step a g-node is removed

In the extended version: *no*



corresponding to rewriting $f(g^\omega)$ to itself

Termination of term graph rewriting is strictly stronger than termination of term rewriting, both in the basic and extended version

Termination of term graph rewriting is strictly stronger than termination of term rewriting, both in the basic and extended version

Example

Termination of term graph rewriting is strictly stronger than termination of term rewriting, both in the basic and extended version

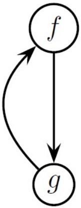
Example

$f(g(x)) \to g(f(x))$ is terminating as a term rewrite system

Termination of term graph rewriting is strictly stronger than termination of term rewriting, both in the basic and extended version

Example

$f(g(x)) \to g(f(x))$ is terminating as a term rewrite system



rewrites to itself both in the basic and extended version, hence is not terminating

For both versions, more general, for any term graph rewrite system (TGRS) given by rules $L \leftarrow I \rightarrow R$, we wonder how to prove termination

For both versions, more general, for any term graph rewrite system (TGRS) given by rules $L \leftarrow I \rightarrow R$, we wonder how to prove termination

For a related notion, namely *graph transformation systems* (GTS) in 2014 and 2015 we developed techniques for proving termination automatically, and implemented this in our tool *Grez*

For both versions, more general, for any term graph rewrite system (TGRS) given by rules $L \leftarrow I \rightarrow R$, we wonder how to prove termination

For a related notion, namely *graph transformation systems* (GTS) in 2014 and 2015 we developed techniques for proving termination automatically, and implemented this in our tool *Grez*

GTS is extensively studied and used; standard semantics also based on double push-out

For both versions, more general, for any term graph rewrite system (TGRS) given by rules $L \leftarrow I \rightarrow R$, we wonder how to prove termination

For a related notion, namely *graph transformation systems* (GTS) in 2014 and 2015 we developed techniques for proving termination automatically, and implemented this in our tool *Grez*

GTS is extensively studied and used; standard semantics also based on double push-out

Main idea: transform TGRS to GTS in such a way that termination of resulting GTS implies termination of original TGRS

Difference between TGRS and GTS

Difference between TGRS and GTS

TG = term graph in TGRS
G = graph in GTS

Difference between TGRS and GTS

TG = term graph in TGRS
G = graph in GTS

- Sometimes TG has a root: not desired for covering cycle rewriting

Difference between TGRS and GTS

TG = term graph in TGRS
G = graph in GTS

- Sometimes TG has a root: not desired for covering cycle rewriting
- In G the edges are labeled, in TG the nodes

Difference between TGRS and GTS

TG = term graph in TGRS
G = graph in GTS

- Sometimes TG has a root: not desired for covering cycle rewriting
- In G the edges are labeled, in TG the nodes
- In G a node may have any number of outgoing edges, in TG there are exactly n numbered outgoing edges where n is the arity of the label of the node

Difference between TGRS and GTS

TG = term graph in TGRS
G = graph in GTS

- Sometimes TG has a root: not desired for covering cycle rewriting
- In G the edges are labeled, in TG the nodes
- In G a node may have any number of outgoing edges, in TG there are exactly n numbered outgoing edges where n is the arity of the label of the node

We have to transform TG to G, and give two ways to do so: the *function encoding* and the *number encoding*

Function encoding

Function encoding

For every symbol f of arity n introduce symbols $f_1, \ldots, f_n$

Function encoding

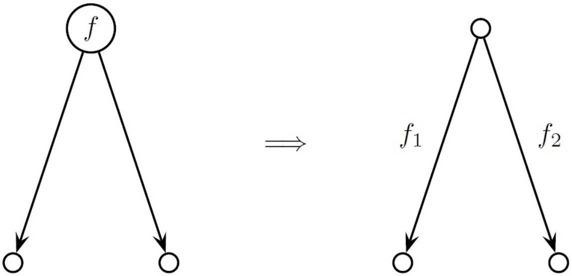
For every symbol f of arity n introduce symbols $f_1, \ldots, f_n$

Give the n numbered outgoing edges of a node labeled by f, new labels $f_1, \ldots, f_n$

Function encoding

For every symbol f of arity n introduce symbols $f_1, \ldots, f_n$

Give the n numbered outgoing edges of a node labeled by f, new labels $f_1, \ldots, f_n$



Number encoding

Number encoding

Keep every symbol f and add new labels $1, \ldots, n$ for n being the highest arity

Number encoding

Keep every symbol f and add new labels $1, \ldots, n$ for n being the highest arity

For every node labeled by f create an edge labeled by f to a fresh node

Number encoding

Keep every symbol f and add new labels $1, \ldots, n$ for n being the highest arity

For every node labeled by f create an edge labeled by f to a fresh node

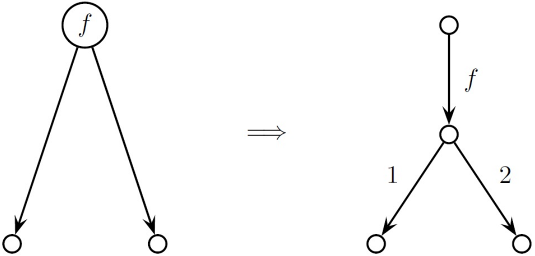
From the fresh node create edges labeled by $1, \ldots, \mathrm{ar}(f)$ to the successor nodes

Number encoding

Keep every symbol f and add new labels $1, \ldots, n$ for n being the highest arity

For every node labeled by f create an edge labeled by f to a fresh node

From the fresh node create edges labeled by $1, \ldots, \mathrm{ar}(f)$ to the successor nodes



Both encodings are sound, that is, termination of transformed GTS implies termination of original TGRS

Both encodings are sound, that is, termination of transformed GTS implies termination of original TGRS

Function encoding is not complete:
$f(a,b) \to f(b,b),\ f(b,a) \to f(a,a)$ is terminating (both basic and extended), but its function encoding is a non-terminating GTS

Both encodings are sound, that is, termination of transformed GTS implies termination of original TGRS

Function encoding is not complete:
$f(a, b) \to f(b, b),\ f(b, a) \to f(a, a)$ is terminating (both basic and extended), but its function encoding is a non-terminating GTS

Number encoding is complete for basic version

Both encodings are sound, that is, termination of transformed GTS implies termination of original TGRS

Function encoding is not complete:
$f(a,b) \to f(b,b),\ f(b,a) \to f(a,a)$ is terminating (both basic and extended), but its function encoding is a non-terminating GTS

Number encoding is complete for basic version

Both have value: there are examples where Grez succeeds for the one and fails for the other, in both directions

Both encodings are sound, that is, termination of transformed GTS implies termination of original TGRS

Function encoding is not complete:
$f(a, b) \to f(b, b)$, $f(b, a) \to f(a, a)$ is terminating (both basic and extended), but its function encoding is a non-terminating GTS

Number encoding is complete for basic version

Both have value: there are examples where Grez succeeds for the one and fails for the other, in both directions

Example requiring heavy techniques from Grez:

$$\begin{array}{rcl} f(x, a(b(y))) & \to & f(c(d(x)), y) \\ f(c(x), y) & \to & f(x, a(y)) \\ f(d(x), y) & \to & f(x, b(y)) \end{array}$$

Surprise: AProVE fails to prove the weaker property of TRS termination

Summary, conclusions

Summary, conclusions

- Term rewriting is a basic framework for computation

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*
- Allowing cycles in term graphs corresponds to *rational terms*: possibly infinite terms, but having finitely many distinct subterms

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*
- Allowing cycles in term graphs corresponds to *rational terms*: possibly infinite terms, but having finitely many distinct subterms
- Term graph rewriting defined in double push-out framework

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*
- Allowing cycles in term graphs corresponds to *rational terms*: possibly infinite terms, but having finitely many distinct subterms
- Term graph rewriting defined in double push-out framework
- Two natural ways to interpret term rewriting on term graphs: basic and extended

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*
- Allowing cycles in term graphs corresponds to *rational terms*: possibly infinite terms, but having finitely many distinct subterms
- Term graph rewriting defined in double push-out framework
- Two natural ways to interpret term rewriting on term graphs: basic and extended
- Proving termination by transforming TGRS to GTS

Summary, conclusions

- Term rewriting is a basic framework for computation
- Sharing is desired for efficiency reasons, by which terms are represented by *term graphs*
- Allowing cycles in term graphs corresponds to *rational terms*: possibly infinite terms, but having finitely many distinct subterms
- Term graph rewriting defined in double push-out framework
- Two natural ways to interpret term rewriting on term graphs: basic and extended
- Proving termination by transforming TGRS to GTS
- Drastically extends set of benchmarks for GTS termination proofs