The complexity of term rewrite systems

Thomas Powell

University of Innsbruck

Proof, Complexity and Verification Seminar
Swansea University
4 December 2014

Outline of talk

Introduction. Describe the general theme.

Part I. Proof theoretic analysis of termination methods

- Recall some well-known results from classical proof theory and rewriting theory.
- Show how they can be brought together to obtain new complexity results.

Part II. Higher-order rewriting and other topics

- Discuss notions of complexity at higher-types.
- Suggest some directions in which progress could be made.

Term rewrite systems

The set of terms $\mathcal{T}(\mathcal{V}, \mathcal{F})$ over some countable set of variables $\mathcal{V}$ and set of function symbols $\mathcal{F}$ is defined by

$$\mathcal{T}(\mathcal{V}, \mathcal{F}) := x \in \mathcal{V} \mid f(t_1, \ldots, t_n) \text{ for } f \in \mathcal{F}$$

A rewrite rule $l \to r$ is a relation between terms such that l is not a variable, and all variables occuring in r also occur in l. A term rewrite system $\mathcal{R}$ is a set of rewrite rules.

The rewrite relation $\to_{\mathcal{R}}$ is the least relation containing $\mathcal{R}$ satisfying

(i) if $s \to_{\mathcal{R}} t$ and σ is a substitution, then $s\sigma \to_{\mathcal{R}} t\sigma$,

(ii) if $s \to_{\mathcal{R}} t$ and $f \in \mathcal{F}$ then $f(\ldots, s, \ldots) \to_{\mathcal{R}} f(\ldots, t, \ldots)$.

We will typically write $\to$ instead of $\to_{\mathcal{R}}$.

Suppose we are given an arbitrary term rewrite system $\mathcal{R}$

$$l_1 \to r_1$$
$$\dots$$
$$l_n \to r_n$$

Important questions

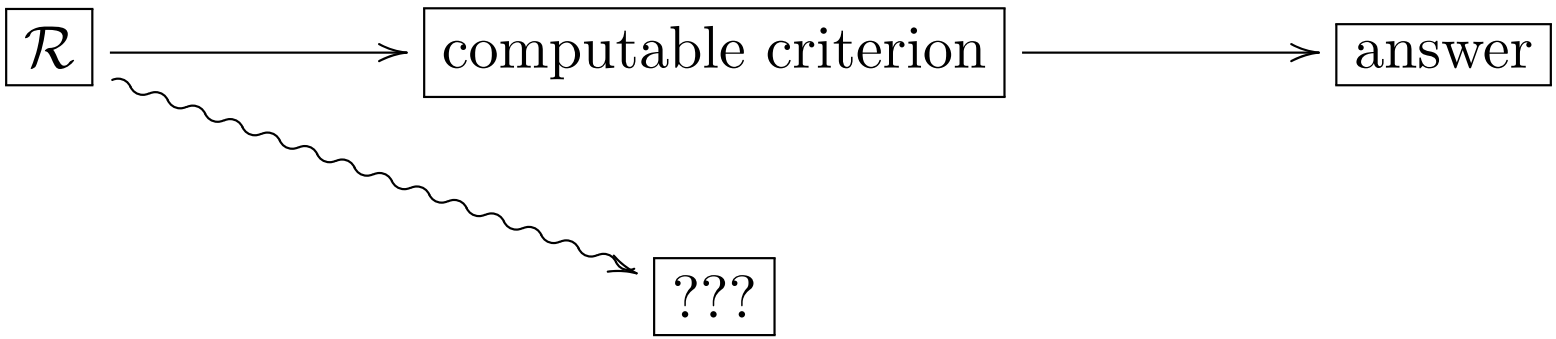
1. Is $\mathcal{R}$ terminating?
2. Is $\mathcal{R}$ confluent?
3. What is the complexity of $\mathcal{R}$?

Suppose we are given an arbitrary term rewrite system $\mathcal{R}$

$$l_1 \to r_1$$
$$\dots$$
$$l_n \to r_n$$

Important questions

1. Is $\mathcal{R}$ terminating?
2. Is $\mathcal{R}$ confluent?
3. What is the complexity of $\mathcal{R}$?



Example: Multiset path order (MPO)

We say that $s <_{\text{mpo}} t = g(t_1, \ldots, t_n)$ relative to some well-founded precedence $\prec$ on function symbols if either

1. $s \leq_{\text{mpo}} t_i$ some $i = 1, \ldots, n$;
2. $s = f(s_1, \ldots, s_m)$ with $f \prec g$ and $s_1, \ldots, s_m <_{\text{mpo}} t$;
3. $s = g(s_1, \ldots, s_n)$, $s_1, \ldots, s_n <_{\text{mpo}} t$ and $(s_1, \ldots, s_n) \ll^{\text{mul}} (t_1, \ldots, t_n)$.

Example: Multiset path order (MPO)

We say that $s <_{\text{mpo}} t = g(t_1, \ldots, t_n)$ relative to some well-founded precedence $\prec$ on function symbols if either

1. $s \leq_{\text{mpo}} t_i$ some $i = 1, \ldots, n$;
2. $s = f(s_1, \ldots, s_m)$ with $f \prec g$ and $s_1, \ldots, s_m <_{\text{mpo}} t$;
3. $s = g(s_1, \ldots, s_n)$, $s_1, \ldots, s_n <_{\text{mpo}} t$ and $(s_1, \ldots, s_n) \ll^{\text{mul}} (t_1, \ldots, t_n)$.

Theorem (Hofbauer)

If the rules of $\mathcal{R}$ are reducing under $<_{\text{mpo}}$, then $\mathcal{R}$ is terminating and its derivational complexity is bounded by some primitive recursive function.

$$\mathcal{R}_1 \begin{cases} +(0,m) \to m \\ +(s(n),m) \to s(+(n,m)) \\ \times(0,m) \to 0 \\ \times(s(n),m) \to +(\times(n,m),m) \end{cases}$$

$\mathcal{R}_1$ is reducing under $<_{\text{mpo}}$ w.r.t. $s \prec + \prec \times$.

$$\mathcal{R}_1 \begin{cases} +(0,m) \to m \\ +(s(n),m) \to s(+(n,m)) \\ \times(0,m) \to 0 \\ \times(s(n),m) \to +(\times(n,m),m) \end{cases}$$

$\mathcal{R}_1$ is reducing under $<_{\text{mpo}}$ w.r.t. $s \prec + \prec \times$.

$$\mathcal{R}_2 \begin{cases} f(0,m) \to g(m) \\ f(s(n),m) \to h(n,m,f(n,m)) \end{cases}$$

$\mathcal{R}_2$ is reducing under $<_{\text{mpo}}$ w.r.t. $g, h \prec f$.

In both examples we use that $(n,m) \ll^{\text{mul}} (s(n),m)$.

Example: Lexicographic path ordering (LPO)

We say that $s <_{\text{lpo}} t = g(t_1, \ldots, t_n)$ relative to some well-founded precedence $\prec$ on function symbols if either

1. $s \leq_{\text{mpo}} t_i$ some $i = 1, \ldots, n$;
2. $s = f(s_1, \ldots, s_m)$ with $f \prec g$ and $s_1, \ldots, s_m <_{\text{mpo}} t$;
3. $s = g(s_1, \ldots, s_n)$, $s_1, \ldots, s_n <_{\text{mpo}} t$ and $(s_1, \ldots, s_n) \ll^{\text{lex}} (t_1, \ldots, t_n)$.

Example: Lexicographic path ordering (LPO)

We say that $s <_{\text{lpo}} t = g(t_1, \ldots, t_n)$ relative to some well-founded precedence $\prec$ on function symbols if either

1. $s \leq_{\text{mpo}} t_i$ some $i = 1, \ldots, n$;
2. $s = f(s_1, \ldots, s_m)$ with $f \prec g$ and $s_1, \ldots, s_m <_{\text{mpo}} t$;
3. $s = g(s_1, \ldots, s_n)$, $s_1, \ldots, s_n <_{\text{mpo}} t$ and $(s_1, \ldots, s_n) \ll^{\text{lex}} (t_1, \ldots, t_n)$.

Theorem (Weiermann)

If the rules of $\mathcal{R}$ are reducing under $<_{\text{lpo}}$, then $\mathcal{R}$ is terminating and its derivational complexity is bounded by some multiply recursive function.

$$\mathcal{R}_3 \left\{ \begin{array}{c} A(0,m) \to s(m) \\ A(s(n),m) \to A(n,1) \\ A(s(n),s(m)) \to A(n,A(s(n),m)) \end{array} \right.$$

$\mathcal{R}_3$ is *not* reducing under $<_{\mathrm{mpo}}$, but since
$(n, A(s(n), m)) \ll^{\mathrm{lex}} (s(n), s(m))$ it is reducing under $\ll^{\mathrm{lex}}$.

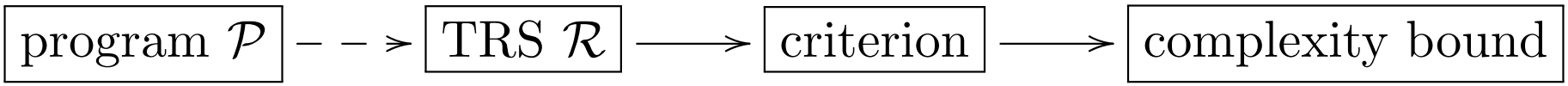
$$\mathcal{R}_3 \begin{cases} A(0,m) \to s(m) \\ A(s(n),m) \to A(n,1) \\ A(s(n),s(m)) \to A(n,A(s(n),m)) \end{cases}$$

$\mathcal{R}_3$ is *not* reducing under $<_{\text{mpo}}$, but since $(n, A(s(n),m)) \ll^{\text{lex}} (s(n),s(m))$ it is reducing under $\ll^{\text{lex}}$.

$$\mathcal{R}_4 \begin{cases} f(0,m) \to g(m) \\ f(s(n),m) \to h(n,m,f(n,p(n,m,f(n,m)))) \end{cases}$$

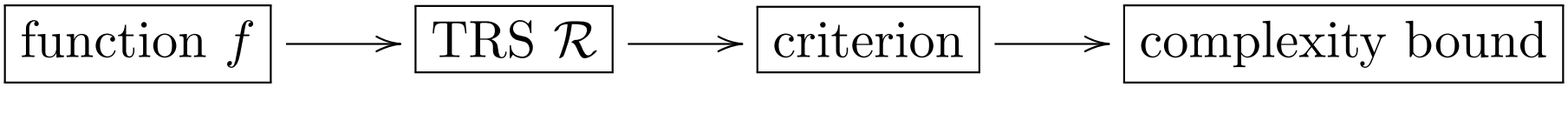
$\mathcal{R}_4$ is only reducing under $<_{\text{lpo}}$, but actually has primitive recursive complexity...

Application: Complexity analysis of programs



- How do we design the translation $\boxed{\mathcal{P}} \to \boxed{\mathcal{R}}$ so that it is complexity preserving?
- For a given programming language, what is the correct abstract representation (constraint rewriting, higher-order...)?
- We want criteria which capture a rich variety of programs, and which guarantee *feasible* computations.

Application: Computability theory



- Can we infer something about the behaviour of a function from its defining equation?
- What about higher-order functionals?
- Here our interest is not restricted to feasible computations, but to all kinds of complexity classes: primitive recursive, multiply recursive and beyond... (bar recursive?)

Part I. Proof theoretic analysis of termination methods

Theorem. For all integers n there exists some prime $p \geq n$.

Implicit bound in Euclid's proof: $p(n) \leq n! + 1$.

Theorem. For all integers n there exists some prime $p \geq n$.

Implicit bound in Euclid's proof: $p(n) \leq n! + 1$.

Theorem. For all terms t in $\mathcal{R}$ there exists some N such that all rewrite sequence starting from t have length $\leq N$.

Can we find a concrete bound in termination proof: $N(t) \leq f(|t|)$?

Theorem. For all integers n there exists some prime $p \geq n$.

Implicit bound in Euclid's proof: $p(n) \leq n! + 1$.

Theorem. For all terms t in $\mathcal{R}$ there exists some N such that all rewrite sequence starting from t have length $\leq N$.

Can we find a concrete bound in termination proof: $N(t) \leq f(|t|)$?

multiset path ordering $\Rightarrow$ primitive recursive d. c.

lexicographic path ordering $\Rightarrow$ multiply recursive d. c.

General challenge:

$$\boxed{\text{termination proof technique}} \Rightarrow \boxed{\text{upper bound on complexity}}$$

G. Kreisel "What more do we know if we have a proved a theorem by restricted means than if we merely know that it is true?"

General challenge:

$$\boxed{\text{termination proof technique}} \Rightarrow \boxed{\text{upper bound on complexity}}$$

G. Kreisel "What more do we know if we have a proved a theorem by restricted means than if we merely know that it is true?"

Techniques such as proof interpretations - which analyse the logical structure of proofs - provide us with a powerful tool for tackling this problem, and have great potential for both:

- Obtaining new complexity results;
- Understanding *why* termination techniques yield the complexity bounds that they do.

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$A \to \forall n \exists m B_0(n, m)$$

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$\begin{aligned} & A \to \forall n \exists m B_0(n,m) \\ & \Rightarrow \forall n (A \to \exists m B_0(n,m)) \end{aligned}$$

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$\begin{aligned} & A \to \forall n \exists m B_0(n,m) \\ & \Rightarrow \forall n (A \to \exists m B_0(n,m)) \\ & \Rightarrow \forall n (\exists \varepsilon \ \mathrm{app}(A_\varepsilon) \to \exists m B_0(n,m)) \end{aligned}$$

where $A \leftrightarrow \exists \varepsilon \ \mathrm{app}(A_\varepsilon)$ for $\mathrm{app}(A_\varepsilon)$ universal.

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$\begin{aligned} & A \to \forall n \exists m B_0(n,m) \\ & \Rightarrow \forall n (A \to \exists m B_0(n,m)) \\ & \Rightarrow \forall n (\exists \varepsilon \ \mathrm{app}(A_\varepsilon) \to \exists m B_0(n,m)) \\ & \Rightarrow \forall \varepsilon, n \exists m (\mathrm{app}(A_\varepsilon) \to B_0(n,m)) \end{aligned}$$

where $A \leftrightarrow \exists \varepsilon \ \mathrm{app}(A_\varepsilon)$ for $\mathrm{app}(A_\varepsilon)$ universal.

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$\begin{aligned}
&A \to \forall n \exists m B_0(n,m) \\
&\Rightarrow \forall n (A \to \exists m B_0(n,m)) \\
&\Rightarrow \forall n (\exists \varepsilon \ \mathrm{app}(A_\varepsilon) \to \exists m B_0(n,m)) \\
&\Rightarrow \forall \varepsilon, n \exists m (\mathrm{app}(A_\varepsilon) \to B_0(n,m)) \\
&\Rightarrow \exists f \forall \varepsilon, n (\mathrm{app}(A_\varepsilon) \to B_0(n, f_\varepsilon(n)))
\end{aligned}$$

where $A \leftrightarrow \exists \varepsilon \ \mathrm{app}(A_\varepsilon)$ for $\mathrm{app}(A_\varepsilon)$ universal.

In one line, a proof interpretation is a formal translation acting on the logical structure of proofs. E.g. Gödel's Dialectica interpretation.

Of particular interest are (combinations of) interpretations that directly realize $\forall\exists$-formulas.

$$\begin{aligned}
&A \to \forall n \exists m B_0(n,m) \\
&\Rightarrow \forall n (A \to \exists m B_0(n,m)) \\
&\Rightarrow \forall n (\exists \varepsilon \ \mathrm{app}(A_\varepsilon) \to \exists m B_0(n,m)) \\
&\Rightarrow \forall \varepsilon, n \exists m (\mathrm{app}(A_\varepsilon) \to B_0(n,m)) \\
&\Rightarrow \exists f \forall \varepsilon, n (\mathrm{app}(A_\varepsilon) \to B_0(n, f_\varepsilon(n)))
\end{aligned}$$

where $A \leftrightarrow \exists \varepsilon \ \mathrm{app}(A_\varepsilon)$ for $\mathrm{app}(A_\varepsilon)$ universal.

The computational complexity of f_ε is determined by the proof-theoretic complexity of A.

Classic results of 20th century proof theory (Gödel 1958, Parsons 1972):

$$\begin{aligned}
\mathsf{PA} \vdash \forall n \exists m A_0(n,m) \Rightarrow & \exists f \in \mathcal{F}_{\varepsilon_0} \forall n A_0(n, f(n)) \\
& f \ \textit{in system} \ \mathsf{T} \\
& \vdots \\
\mathsf{I}\Sigma_2 \vdash \forall n \exists m A_0(n,m) \Rightarrow & \exists f \in \mathcal{F}_{\omega^\omega} \forall n A_0(n, f(n)) \\
& f \ \textit{multiply recursive} \\
\mathsf{I}\Sigma_1 \vdash \forall n \exists m A_0(n,m) \Rightarrow & \exists f \in \mathcal{F}_{\omega} \forall n A_0(n, f(n)) \\
& f \ \textit{primitive recursive}
\end{aligned}$$

Application to program complexity (Buchholz 1994):

Step 1. Any finite rewrite system $\mathcal{R}$ reducing under $<_{\text{mpo}}$ or $<_{\text{lpo}}$ uses only a (finitely branching) approximation $<^k_{\text{mpo}}$, $<^k_{\text{lpo}}$ of these orderings, where k depends on $\mathcal{R}$.

Step 2. $<^k_{\text{mpo}}$ is provably well-founded in $\mathsf{I}\Sigma_1$, while $<^k_{\text{lpo}}$ is provably well-founded in $\mathsf{I}\Sigma_2$.

Application to program complexity (Buchholz 1994):

Step 1. Any finite rewrite system $\mathcal{R}$ reducing under $<_{\text{mpo}}$ or $<_{\text{lpo}}$ uses only a (finitely branching) approximation $<^k_{\text{mpo}}$, $<^k_{\text{lpo}}$ of these orderings, where k depends on $\mathcal{R}$.

Step 2. $<^k_{\text{mpo}}$ is provably well-founded in $\mathsf{I}\Sigma_1$, while $<^k_{\text{lpo}}$ is provably well-founded in $\mathsf{I}\Sigma_2$.

Result.

$$\text{multiset path ordering } (\mathsf{I}\Sigma_1) \Rightarrow N(t) \leq f(|t|) \text{ for } f \in \mathcal{F}_\omega$$
$$\text{lexicographic path ordering } (\mathsf{I}\Sigma_2) \Rightarrow N(t) \leq f(|t|) \text{ for } f \in \mathcal{F}_{\omega^\omega}$$

Very rough explanation...

Want to prove $t_1, \ldots, t_n$ well-founded w.r.t. $<$ then $(t_1, \ldots, t_n)$ well-founded w.r.t. $\ll^{\text{mul}}/\ll^{\text{lex}}$.

$$(t_1, \ldots, t_{j-1}, t'_1, \ldots, t'_m, t_{j+1}, \ldots, t_n) \ll^{\text{mul}} (t_1, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_n)$$

$$(t_1, \ldots, t_{j-1}, t'_j, \underbrace{s_{j+1}, \ldots, s_n}_{\text{'uncontrolled'}}) \ll^{\text{lex}} (t_1, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_n)$$

Very rough explanation...

Want to prove $t_1, \ldots, t_n$ well-founded w.r.t. $<$ then $(t_1, \ldots, t_n)$ well-founded w.r.t. $\ll^{\text{mul}}/\ll^{\text{lex}}$.

$$(t_1, \ldots, t_{j-1}, t'_1, \ldots, t'_m, t_{j+1}, \ldots, t_n) \ll^{\text{mul}} (t_1, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_n)$$

$$(t_1, \ldots, t_{j-1}, t'_j, \underbrace{s_{j+1}, \ldots, s_n}_{\text{'uncontrolled'}}) \ll^{\text{lex}} (t_1, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_n)$$

Need a universal quantification to deal with uncontrolled terms in $\ll^{\text{lex}}$:

$$B(t_{j-1}) :\equiv \forall s_{j+1}, \ldots, s_n[(t_1, \ldots, t_{j-1}, t'_j, s_{j+1}, \ldots, s_n) \text{ w.f.}]$$

Causes a one-step shift up the arithmetic hierarchy!

The last few decades has seen great advances in proof theoretic techniques and their application in mathematics and computer science.

In particular, meta-theorems of the form

$$\mathcal{T} \vdash A \Rightarrow \exists t \in \mathcal{F}(t \text{ interprets } A)$$

have become increasingly refined and sophisticated, and oriented towards practical as opposed to foundational results.

The last few decades has seen great advances in proof theoretic techniques and their application in mathematics and computer science.

In particular, meta-theorems of the form

$$\mathcal{T} \vdash A \Rightarrow \exists t \in \mathcal{F}(t \text{ interprets } A)$$

have become increasingly refined and sophisticated, and oriented towards practical as opposed to foundational results.

QUESTION:

Can we develop new and interesting meta-theorems (along the lines of Buchholz '94, but not necessarily restricted to path orders!) which capture families of termination proofs and can be used to derive corresponding complexity bounds in a uniform way?

A generalised path order

Suppose that $\ll$ is some arbitrary extension to tuples of an ordering $<$ on terms. Let $<_{\ll}$ be the corresonding path order built up inductively from $\ll$, and $<^k_{\ll}$ its k-approximation.

A generalised path order

Suppose that $\ll$ is some arbitrary extension to tuples of an ordering $<$ on terms. Let $<_{\ll}$ be the corresonding path order built up inductively from $\ll$, and $<^k_{\ll}$ its k-approximation.

Proposition

If well-foundedness of $\ll$ is provable in $\mathsf{I}\Sigma_n$, then well-foundedness of $<^k_{\ll}$ is provable in $\mathsf{I}\Sigma_n$.

Corollary

If the TRS $\mathcal{R}$ is terminating by $<_{\ll}$, then it is terminating by $<^k_{\ll}$ for some k, and therefore $N(t) \leq f(|t|)$ for some $f \in \mathcal{F}_{\omega_n}$.

In fact, with a little effort, we can extract an explicit term $\Phi^k(t)$ witnessing well-foundedness of $<^k_{\ll}$ from which we can construct f.

Let $\Phi^k : \mathcal{T} \to \mathcal{T}^*$ be defined by $\Phi^k(x) = []$ for variables x, and

$$\Phi^k(g(\underline{t})) =$$

$$\left(\bigoplus_{s <^k g(\underline{t})} \begin{cases} \Phi(t_i) & \text{if } s \leq^k t_i \\ \Phi_3(f, \underline{s}, \bar{\Phi}_1(g, \underline{t}, \Phi(\underline{t}), \Phi_2(g, \Phi_3|_{f \prec g})|_{\underline{s} \in \underline{d'} \wedge \underline{s} \ll_g \underline{t}}, \Phi_3|_{f \prec g}, \underline{s})) & \text{if } s = f(\underline{s}) \wedge f \prec g \\ \Phi_2(g, \underline{s}, \bar{\Phi}_1(g, \underline{t}, \Phi(\underline{t}), \Phi_2(g, \Phi_3|_{f \prec g})|_{\underline{s} \in \underline{d'} \wedge \underline{s} \ll_g \underline{t}}, \Phi_3|_{f \prec g}, \underline{s}), \Phi_3|_{f \prec g}) & \text{if } s = g(\underline{s}) \end{cases} \right)$$

$$* \, [g(\underline{t})]$$

where each of Φ_i for $i = 1, 2, 3$ are recursively defined elements of T_{n-1} (details omitted!). Then for all terms t, if $s \leq^k_{\ll} t$ then $s \in \Phi^k(t)$.

Let $\Phi^k : \mathcal{T} \to \mathcal{T}^*$ be defined by $\Phi^k(x) = []$ for variables x, and

$$\Phi^k(g(\underline{t})) =$$
$$\left(\bigoplus_{s <^k g(\underline{t})} \begin{cases} \Phi(t_i) & \text{if } s \leq^k t_i \\ \Phi_3(f, \underline{s}, \bar{\Phi}_1(g, \underline{t}, \Phi(\underline{t}), \Phi_2(g, \Phi_3|_{f \prec g})|_{\underline{s} \in \underline{d'} \wedge \underline{s} \ll_g \underline{t}}, \Phi_3|_{f \prec g}, \underline{s})) & \text{if } s = f(\underline{s}) \wedge f \prec g \\ \Phi_2(g, \underline{s}, \bar{\Phi}_1(g, \underline{t}, \Phi(\underline{t}), \Phi_2(g, \Phi_3|_{f \prec g})|_{\underline{s} \in \underline{d'} \wedge \underline{s} \ll_g \underline{t}}, \Phi_3|_{f \prec g}, \underline{s}), \Phi_3|_{f \prec g}) & \text{if } s = g(\underline{s}) \end{cases} \right)$$
$$* [g(\underline{t})]$$

where each of Φ_i for $i = 1, 2, 3$ are recursively defined elements of T_{n-1} (details omitted!). Then for all terms t, if $s \leq^k_{\ll} t$ then $s \in \Phi^k(t)$.

Unsurprisingly, extracted term quite complex! But we have something concrete to analyse: Crucial point is that exact parameters of the path order, along with any additional restrictions imposed, will result in corresponding changes to the term Φ.

Suppose that well-foundedness of $\ll$ is provable in $\mathsf{I}\Sigma_1$. We already know that if $\mathcal{R}$ is terminating by $<_{\ll}$ then $N(t) \leq f(t)$ for some primitive recursive f.

What if there are additional restrictions on $\mathcal{R}$? e.g.

- $\max_{f \in F}(\text{arity}(f)) \leq k$;
- $|F| \leq k$;
- $g(\underline{t})$ with $\underline{t} \in S^n$...

Will these guarantee that $f \in \mathcal{A}_m$ for some m? Can we design new criteria which induce feasible derivational complexity?

It would be very interesting to devise extensions $\ll$ of the multiset ordering such that

- well-foundedness of $\ll$ is provable in $\mathsf{I}\Sigma_1$,
- $<_{\ll}$ admits new TRSs $\mathcal{R}$ not admitted by $\ll^{\mathrm{mul}}$ e.g. single nested recursion:

$$\begin{aligned} f(0, n) &\to g(n) \\ f(s(n), m) &\to h(n, m, f(n, p(n, m, f(n, m)))) \end{aligned}$$

It would be very interesting to devise extensions $\ll$ of the multiset ordering such that

- well-foundedness of $\ll$ is provable in $\mathsf{I}\Sigma_1$,
- $<_{\ll}$ admits new TRSs $\mathcal{R}$ not admitted by $\ll^{\mathrm{mul}}$ e.g. single nested recursion:

$$f(0, n) \to g(n)$$
$$f(s(n), m) \to h(n, m, f(n, p(n, m, f(n, m))))$$

This would enlarge the class of primitive recursive schemata which could be automatically verified as primitive recursive, and both reobtain and extend classic results of recursion theory.

Idea originally due to A. Weiermann (unpublished), who uses ordinal analysis to verify complexity. Proof interpretations potentially offer a smoother means of extending path orders and verifying complexity bound.

Method bears close connections with alternative approaches that deal explicitly with ordinal analysis or monotone assingments.

Nevertheless, focusing on the precise formalization of termination arguments and appealing to proof-theoretic meta-theorems has several advantages.

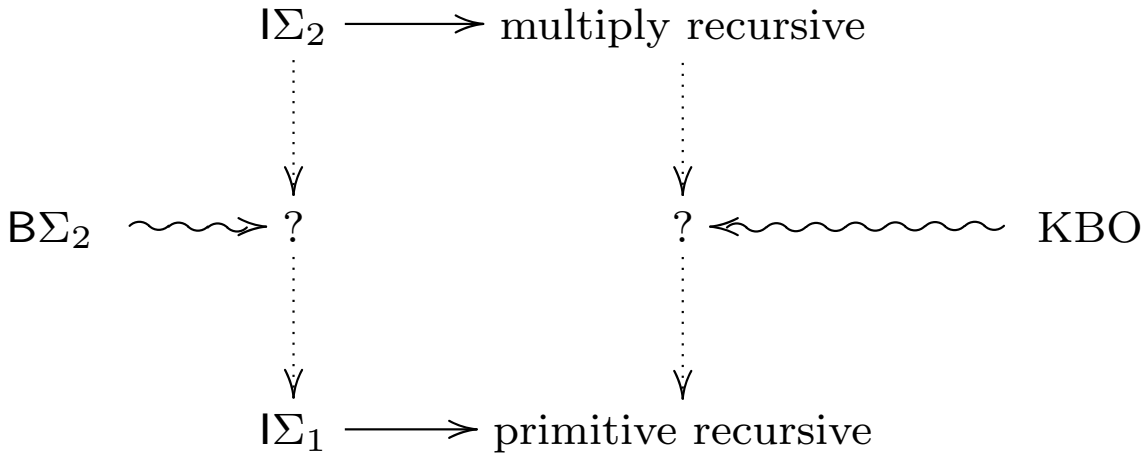
Method bears close connections with alternative approaches that deal explicitly with ordinal analysis or monotone assingments.

Nevertheless, focusing on the precise formalization of termination arguments and appealing to proof-theoretic meta-theorems has several advantages.

- Reveals on a deep, structural level *why* a termination proofs produces a certain complexity bound, and encourages a uniform and general way of thinking about these proofs.
- Great potential for new applications in the analysis of a wider range of termination methods, building on substantial amount of work done in last 20 years.
- Can simultaneously work towards new complexity bounds while developing broadly applicable results in proof theory.

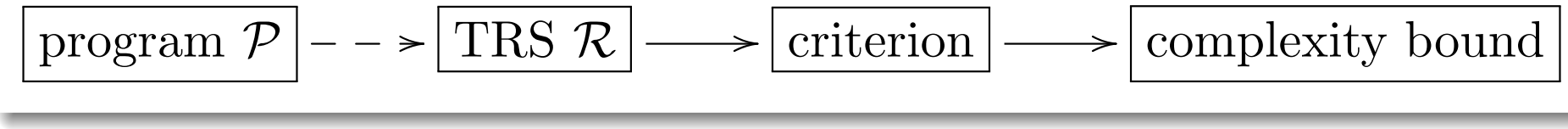
Example

Most existing applications of proof theory focus on obtaining primitive recursive bounds. On the other hand, many termination methods induce multiply recursive complexity.

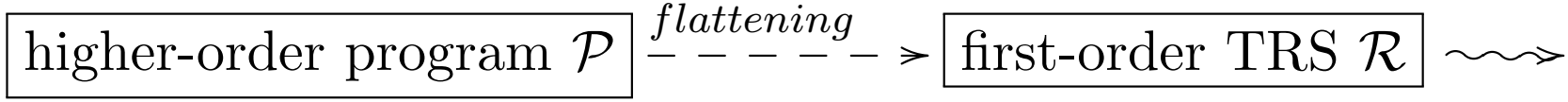


Can Parsons' result be extended in an interesting way to systems between $\mathsf{I}\Sigma_1$ and $\mathsf{I}\Sigma_2$?

Part II. Higher-order rewriting and other topics



What if $\mathcal{P}$ is a higher-type program? For termination, the following approach has been shown successful:



However, de-functionalisation is not generally complexity preserving, so technique does not seen to work for complexity analysis.

Question

How can we determine the complexity of higher-order term rewrite systems?

Question

How can we determine the complexity of higher-order term rewrite systems?

Problem: What do we even mean by the complexity of a higher-order term?

$$\mathsf{map} \colon (\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$$

$$\begin{aligned} \mathsf{map}\ f\ [] &\to [] \\ \mathsf{map}\ f\ (x : xs) &\to (f\ x) : (\mathsf{map}\ f\ xs) \end{aligned}$$

$$\mathsf{fold} \colon \mathsf{Nat} \to (\mathsf{Nat} \to \mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$$

$$\begin{aligned} \mathsf{fold}\ a\ f\ [] &\to a \\ \mathsf{fold}\ a\ f\ (x : xs) &\to f\ x\ (\mathsf{fold}\ a\ f\ xs) \end{aligned}$$

Higher-order polynomial interpretations (Baillot, Dal-Lago 2012)

The functional map is interpreted as a higher-type polynomial $[\mathsf{map}] : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$. It is shown that whenever $[f] : \mathbb{N} \to \mathbb{N}$ has a polynomial interpretation, $\mathsf{map}\ f : \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$ is a polynomial time program.

A more intricate analysis deals with fold - if f has a polynomial interpretation *and* satisfies an additional termination criterion (i.e. size restriction), then $\mathsf{fold}\ a\ f : \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$ a polynomial time program.

Higher-order polynomial interpretations (Baillot, Dal-Lago 2012)

The functional map is interpreted as a higher-type polynomial $[\mathsf{map}] : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N}$. It is shown that whenever $[f] : \mathbb{N} \to \mathbb{N}$ has a polynomial interpretation, $\mathsf{map}\ f : \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$ is a polynomial time program.

A more intricate analysis deals with fold - if f has a polynomial interpretation *and* satisfies an additional termination criterion (i.e. size restriction), then $\mathsf{fold}\ a\ f : \mathsf{List}(\mathsf{Nat}) \to \mathsf{List}(\mathsf{Nat})$ a polynomial time program.

Summary: Certain higher-order functionals yield polynomial time programs whenever their function arguments are of a suitably restricted complexity.

Static cost analysis (Danner, Paykin, Royer 2013)

A compositional complexity measure $\|t\|$ is assigned to higher-type functionals, based on $\langle \text{cost}, \text{potential} \rangle$ pairs - the cost of evaluating t together with the cost of using t. For example, the cost of $\lambda x.t \colon \mathsf{Nat} \to \mathsf{Nat}$ is 1, but its potential describes the complexity of $t[x \to \mathbf{n}]$ for all possible numerals.

$$\mathsf{map}_c(f, 0) = 0 \quad \mathsf{map}_c(f, n+1) = (f_p(1))_c + \mathsf{map}_c(f, n)$$
$$\mathsf{map}_p(f, 0) = 0 \quad \mathsf{map}_p(f, n+1) = 1 + \mathsf{map}_p(f, n)$$

Static cost analysis (Danner, Paykin, Royer 2013)

A compositional complexity measure $\|t\|$ is assigned to higher-type functionals, based on $\langle \text{cost}, \text{potential} \rangle$ pairs - the cost of evaluating t together with the cost of using t. For example, the cost of $\lambda x.t\colon \mathsf{Nat} \to \mathsf{Nat}$ is 1, but its potential describes the complexity of $t[x \to \mathbf{n}]$ for all possible numerals.

$$\mathsf{map}_c(f, 0) = 0 \quad \mathsf{map}_c(f, n+1) = (f_p(1))_c + \mathsf{map}_c(f, n)$$
$$\mathsf{map}_p(f, 0) = 0 \quad \mathsf{map}_p(f, n+1) = 1 + \mathsf{map}_p(f, n)$$

Summary: The complexity of any program t in system $\mathcal{T}$ is defined by induction over the structure of t. Further analysis the resulting recursive equations for complexity can produce an upper bound for arbitrary input e.g.

$$\mathsf{map}_c(f, n) = n \cdot (f_p(1))_c.$$

Question

Can we build on these ideas and provide some kind of automated complexity analysis that captures a rich variety of programs?

- Jounnaud et al. have generalised recursive path orders to higher-order rewrite systems? Can these be refined to provide an upper bound on complexity as in the first order case?
- A powerful method of automated *termination* analysis (e.g. Giesl et al.) is to examine so called 'symbolic evaluation graphs', which represent all possible evaluations of a program in a finite way. Could a similar technique work for complexity?
- But perhaps a completely new approach is needed...

Conclusion

Powerful criteria for capturing the complexity of abstract models of computational such as rewrite systems would be of significance in both

- the automated complexity analysis of programs,
- establishing e.g. closure properties in computability theory.

There are a wealth of techniques from

- classic proof theory,
- traditional (first-order) rewriting theory

that could be potentially adapted for this purpose.