On the computational content of termination proofs

Georg Moser and Thomas Powell

University of Innsbruck

Computability in Europe 2015
Bucharest, Romania
2 July 2015

Overview

The proceedings paper is intended as a small illustratation of an idea which we hope to develop properly in the future. Significance not so much in the results, but in the method.

Main purpose of the talk will provide background in order to motivate the main topic of paper.

1. Extracting computational content from proofs.
2. Analysing the complexity of programs.
3. Brief sketch of results.
4. The next step - goals for the future.

1. Extracting computational content from proofs.

Theorem. For all $n \in \mathbb{N}$ there exists some $p \geq n$ such that $\mathsf{prime}(p)$.

Proof (Euclid). Suppose that $p_1, \ldots, p_m$ are the primes less than n. Then $p_1 \ldots p_m + 1$ has a prime factor p distinct from the p_i, and so in particular $p \geq n$.

Theorem. For all $n \in \mathbb{N}$ there exists some $p \geq n$ such that $\mathsf{prime}(p)$.

Proof (Euclid). Suppose that $p_1, \ldots, p_m$ are the primes less than n. Then $p_1 \ldots p_m + 1$ has a prime factor p distinct from the p_i, and so in particular $p \geq n$.

A computational perspective

Theorem = Specification:

$$\underbrace{\forall n \in \mathbb{N}}_{\text{input}}\ \underbrace{\exists p \in \mathbb{N}}_{\text{output}}\ \underbrace{(p \geq n \wedge \mathsf{prime}(p))}_{\text{specification}}$$

Proof $\rightsquigarrow$ Algorithm:

$$\mathtt{n} \mapsto \mathtt{least\ prime\ factor\ of\ p_1 \cdot \ldots \cdot p_m + 1}$$

Suppose that Γ is a derivation of

$$\forall x \in X \exists y \in Y \, P(x, y).$$

Then we can extract, recursively over the structure of Γ, a program $f_\Gamma \colon X \to Y$ satisfying

$$\forall x \; P(x, f_\Gamma(x)).$$

Suppose that Γ is a derivation of

$$\forall x \in X \exists y \in Y \, P(x, y).$$

Then we can extract, recursively over the structure of Γ, a program $f_\Gamma \colon X \to Y$ satisfying

$$\forall x \; P(x, f_\Gamma(x)).$$

We expect a relationship between Γ and f_Γ:

Γ	f_Γ
proof-theoretic structure	algorithmic structure
logical complexity	computational complexity

A formal extraction $\Gamma \mapsto f_\Gamma$, even in the case where Γ is non-constructive, can be carried out using proof theoretic tools such as Kreisel's modified realizability, Gödel's Dialectica interpretation.

2. Analysing the complexity of programs.

Term rewrite systems: An abstract model of computation

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 finite 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 $\to_{\mathcal{R}}$.

Example 1. Function symbols 0, s, $+$ and $\times$.

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

Example 1. Function symbols 0, s, $+$ and $\times$.

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

$$s(s(0) + s(0)) \times s(0) \to s(s(0 + s(0))) \times s(0) \to \ldots \to s(s(s(0)))$$

Example 1. Function symbols 0, s, $+$ and $\times$.

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

$$s(s(0) + s(0)) \times s(0) \to s(s(0 + s(0))) \times s(0) \to \ldots \to s(s(s(0)))$$

Example 2. Function symbols 0, s and A.

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

Example 1. Function symbols 0, s, $+$ and $\times$.

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

$$s(s(0) + s(0)) \times s(0) \to s(s(0 + s(0))) \times s(0) \to \ldots \to s(s(s(0)))$$

Example 2. Function symbols 0, s and A.

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

$$A(s(0), s(0)) \to A(0, A(s(0), 0)) \to s(A(s(0), 0)) \to \ldots \to s(s(s(0)))$$

Definition. A tree T is a derivation tree for s if it has s as a root, and for each $t \in T$, the children $\{t_1, \ldots, t_k\}$ of t are those elements such that $t \to t_i$. Branches $s, s_1, \ldots, s_n$ of T are precisely rewrite sequences $s \to s_1 \to \ldots \to s_n$.

Definition. A tree T is a derivation tree for s if it has s as a root, and for each $t \in T$, the children $\{t_1, \ldots, t_k\}$ of t are those elements such that $t \to t_i$. Branches $s, s_1, \ldots, s_n$ of T are precisely rewrite sequences $s \to s_1 \to \ldots \to s_n$.

Definition (Termination A). A term rewrite system is terminating if for all terms s, there are no infinite derivations $s \to s_1 \to s_2 \to \ldots$ starting from s.

Definition. A tree T is a derivation tree for s if it has s as a root, and for each $t \in T$, the children $\{t_1, \ldots, t_k\}$ of t are those elements such that $t \to t_i$. Branches $s, s_1, \ldots, s_n$ of T are precisely rewrite sequences $s \to s_1 \to \ldots \to s_n$.

Definition (Termination A). A term rewrite system is terminating if for all terms s, there are no infinite derivations $s \to s_1 \to s_2 \to \ldots$ starting from s.

All derivation trees are finitely branching, and so termination implies that all derivation trees must be finite (and vice-versa).

Definition (Termination B). A term rewrite system is terminating if for all terms s there exists a finite derivation tree T for s.

Definition (Derivation height). The derivation height $\mathrm{dh}(s)$ of a term s is the maximum length of derivations starting from s i.e.

$$\mathrm{dh}(s) := \max\{n \mid \exists t(s \to^n t)\}.$$

If T is a derivation tree for s, the $\mathrm{dh}(s) = |T|$ where $|T|$ is the length of the maximum branch in T.

Definition (Derivation height). The derivation height $\mathrm{dh}(s)$ of a term s is the maximum length of derivations starting from s i.e.

$$\mathrm{dh}(s) := \max\{n \mid \exists t(s \to^n t)\}.$$

If T is a derivation tree for s, the $\mathrm{dh}(s) = |T|$ where $|T|$ is the length of the maximum branch in T.

Definition (Derivational complexity). The derivational complexity function $\mathrm{dc}\colon \mathbb{N} \to \mathbb{N}$ of a term rewrite system is

$$\mathrm{dc}(n) := \max_{|s| \leq n} \mathrm{dh}(s)$$

where $|s|$ denotes the size of the term s. Equivalently

$$\mathrm{dc}(n) := \max\{|T| \mid T \text{ a derivation tree for some } s \text{ with } |s| \leq n\}.$$

Termination Theorem = Specification:

$$\underbrace{\forall s \in \mathsf{Term}}_{\text{input}}\ \underbrace{\exists T \in \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]}_{\text{output}}\ \underbrace{(T \text{ a derivation tree for } s)}_{\text{specification}}$$

Proof $\rightsquigarrow$ Algorithm of type $\mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$:

$$\mathtt{s} \mapsto \mathtt{T}_{\mathtt{s}}$$

Termination Theorem = Specification:

$$\underbrace{\forall s \in \mathsf{Term}}_{\text{input}}\ \underbrace{\exists T \in \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]}_{\text{output}}\ \underbrace{(T \text{ a derivation tree for } s)}_{\text{specification}}$$

Proof $\rightsquigarrow$ Algorithm of type $\mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$:

$$\mathtt{s} \mapsto \mathtt{T}_{\mathtt{s}}$$

Key Observation: Suppose that Γ is a proof that a TRS terminates. Then we can formally extract a function $f_\Gamma \colon \mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ from Γ such that

$$\forall s(f_\Gamma(s) \text{ a derivation tree for } s)$$

Then from f_Γ we can read of the complexity of $\mathcal{R}$:

$$\mathrm{dc}(n) = \max_{|s| \leq n} \mathrm{dh}(s) = \max_{|s| \leq n} |f_\Gamma(s)|$$

To adapt famous quote of Kreisel:

> *If we have proved termination of a program by restricted means, what can we infer about its complexity?*

Basic idea is that the weaker the proof that a program $\mathcal{P}$ terminates, the stronger the bound on the complexity of $\mathcal{P}$.

This idea is by no means new, but existing complexity analyses of termination proofs typically involve complex combinatorial calculations, that are difficult to extend/generalise.

To adapt famous quote of Kreisel:

> *If we have proved termination of a program by restricted means, what can we infer about its complexity?*

Basic idea is that the weaker the proof that a program $\mathcal{P}$ terminates, the stronger the bound on the complexity of $\mathcal{P}$.

This idea is by no means new, but existing complexity analyses of termination proofs typically involve complex combinatorial calculations, that are difficult to extend/generalise.

Challenge. Carry out a proof-theoretic analysis and *formal term extraction* from termination critera, in order to obtain a bound on the complexity of programs shown to terminate this way, and encourage a uniform way of thinking about the relationship between termination and complexity.

3. Brief sketch of results.

Recursive path orders

Given a well-founded precedence $\succ$ on the set of function symbols $\mathcal{F}$, the recursive path order $\succ_{\text{rpo}}$ on terms with respect to some extension τ to tuples is inductively defined by $t = f(t_1, \ldots, t_n) \succ_{\text{rpo}} s$ if:

1. $t_i \succeq_{\text{rpo}} s$ for some $i = 1, \ldots, n$;
2. $s = g(s_1, \ldots s_m)$ with $f \succ g$ and $t \succ_{\text{rpo}} s_i$ for all $i = 1, \ldots, m$;
3. $s = f(s_1, \ldots, s_n)$, $t \succ_{\text{rpo}} s_i$ for all $i = 1, \ldots, n$ and $\langle t_1, \ldots, t_n \rangle \succ_{\text{rpo},\tau} \langle s_1, \ldots, s_n \rangle$.

Recursive path orders

Given a well-founded precedence $\succ$ on the set of function symbols $\mathcal{F}$, the recursive path order $\succ_{\text{rpo}}$ on terms with respect to some extension τ to tuples is inductively defined by $t = f(t_1, \ldots, t_n) \succ_{\text{rpo}} s$ if:

1. $t_i \succeq_{\text{rpo}} s$ for some $i = 1, \ldots, n$;
2. $s = g(s_1, \ldots s_m)$ with $f \succ g$ and $t \succ_{\text{rpo}} s_i$ for all $i = 1, \ldots, m$;
3. $s = f(s_1, \ldots, s_n)$, $t \succ_{\text{rpo}} s_i$ for all $i = 1, \ldots, n$ and $\langle t_1, \ldots, t_n \rangle \succ_{\text{rpo},\tau} \langle s_1, \ldots, s_n \rangle$.

Important examples

- When τ is the multiset extension then $\succ_{\text{rpo}}$ is the *multiset path order*, denoted by $\succ_{\text{mpo}}$.
- When τ is the lexicographic extension then $\succ_{\text{rpo}}$ is the *multiset path order*, denoted by $\succ_{\text{lpo}}$.

Termination theorem. Both $\succ_{\text{mpo}}$ and $\succ_{\text{lpo}}$ are well-founded, and closed under substitutions and contexts. Therefore if the rewrite rules of $\mathcal{R}$ are contained in either $\succ_{\text{mpo}}$ or $\succ_{\text{lpo}}$, then $\mathcal{R}$ terminates.

Termination theorem. Both $\succ_{\text{mpo}}$ and $\succ_{\text{lpo}}$ are well-founded, and closed under substitutions and contexts. Therefore if the rewrite rules of $\mathcal{R}$ are contained in either $\succ_{\text{mpo}}$ or $\succ_{\text{lpo}}$, then $\mathcal{R}$ terminates.

Example 1. Using the precedence $\times \succ + \succ s, 0$ we can prove that all rules of $\mathcal{R}_1$ are reducing under $\succ_{\text{mpo}}$ e.g.

$$s(n) \times m \succ_{\text{mpo}} (n \times m) + m$$

using clause (2) on $\times \succ +$, since $s(n) \times m \succ_{\text{mpo}} n \times m$ by clause (3).

Termination theorem. Both $\succ_{\text{mpo}}$ and $\succ_{\text{lpo}}$ are well-founded, and closed under substitutions and contexts. Therefore if the rewrite rules of $\mathcal{R}$ are contained in either $\succ_{\text{mpo}}$ or $\succ_{\text{lpo}}$, then $\mathcal{R}$ terminates.

Example 1. Using the precedence $\times \succ + \succ s, 0$ we can prove that all rules of $\mathcal{R}_1$ are reducing under $\succ_{\text{mpo}}$ e.g.

$$s(n) \times m \succ_{\text{mpo}} (n \times m) + m$$

using clause (2) on $\times \succ +$, since $s(n) \times m \succ_{\text{mpo}} n \times m$ by clause (3).

Example 2. Using the precedence $A \succ s, 0$ we can prove that all rules of $\mathcal{R}_2$ are reducing under $\succ_{\text{lpo}}$ e.g.

$$A(s(m), s(n)) \succ_{\text{lpo}} A(m, A(s(m), n)) \tag{1}$$

using clause (3), since $s(m) \succ_{\text{lpo}} m$ and $A(s(m), s(m)) \succ_{\text{lpo}} A(s(m), n)$, again by clause (3).

Summary of paper

From a careful analysis of the standard proofs Γ_1, Γ_2 that $\succ_{\text{mpo}}$, $\succ_{\text{lpo}}$ terminate we extract *explicit* functions $f_{\Gamma_1}, f_{\Gamma_2} : \mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ in System T which compute derivation trees for arbitrary terms s. A simple examination of these terms yields

- f_{Γ_1} is primitive recursive, and therefore so is the derivation complexity of $\mathcal{R}$ whenever it is contained in $\succ_{\text{mpo}}$.
- f_{Γ_2} is multiply recursive, and therefore so is the derivation complexity of $\mathcal{R}$ whenever it is contained in $\succ_{\text{lpo}}$.

Summary of paper

From a careful analysis of the standard proofs Γ_1, Γ_2 that $\succ_{\text{mpo}}$, $\succ_{\text{lpo}}$ terminate we extract *explicit* functions $f_{\Gamma_1}, f_{\Gamma_2} : \mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ in System T which compute derivation trees for arbitrary terms s. A simple examination of these terms yields

- f_{Γ_1} is primitive recursive, and therefore so is the derivation complexity of $\mathcal{R}$ whenever it is contained in $\succ_{\text{mpo}}$.
- f_{Γ_2} is multiply recursive, and therefore so is the derivation complexity of $\mathcal{R}$ whenever it is contained in $\succ_{\text{lpo}}$.

Remark

Basic results are not new: are originally due to Hofbauer (1992), Weiermann (1995) respectively. Morover, a derivation of both is given by Buchholz (1995) using clever calibration of the induction required and Parsons (1970).

Purpose of the results is illustration of a new method. First time that an explicit term $\mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ has been formally extracted.
Key benefits:

Purpose of the results is illustration of a new method. First time that an explicit term $\mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ has been formally extracted.
Key benefits:

- Rederivation of the complexity results completely uniform - because the structure of the terms f_{Γ_1} and f_{Γ_2} reflect the structure of the proofs, they differ only in one subterm corresponding to multiset and lexicographic recursion respectively.

Purpose of the results is illustration of a new method. First time that an explicit term $\mathsf{Term} \to \mathsf{Tree}_{\mathsf{fin}}[\mathsf{Term}]$ has been formally extracted. Key benefits:

- Rederivation of the complexity results completely uniform - because the structure of the terms f_{Γ_1} and f_{Γ_2} reflect the structure of the proofs, they differ only in one subterm corresponding to multiset and lexicographic recursion respectively.
- We provide a concrete bridge between logical strength of the termination proof and complexity of programs. Our method should be robust enough to be extended to more interesting path orders that capture interesting primitive recursive closure properties e.g.

$$\begin{aligned} F(0,n) &\to G(n) \\ F(s(m),n) &\to H(m,n,F(m,p(m,n))). \end{aligned}$$

4. The next step - Goals for the future.

The work presented in the paper constitutes a very small step towards a much bigger set of goals.

- A refinement of existing complexity results e.g. how far can we extend $\succ_{\text{mpo}}$ so that it incorporates a wider class of rewrite systems but does not extend class of primitive recursive functions?
- A wider proof-theoretic analysis of termination proofs (cf. recent work of Berardi et. al 2015 on Podelski-Rybalchenko theorem).
- The development of new criteria for automatically inferring complexity bounds for programs, especially at the level of feasible complexity.
- Extension to complexity at higher types.

Thank you!