A New Term Rewriting Characterisation of ETIME functions

Martin Avanzini and Naohi Eguchi

Institute of Computer Science
University of Innsbruck
Austria

April 5, 2014, DICE 2014, Grenoble

Introduction 1/2

The class of primitive recursive functions is closed under several non-trivial recursion schemata (R. Péter, 1967), e.g.:

- Primitive recursion with parameter substitution (PRP):
  (PRP) $f(x+1, y) = h(x, y, f(x, p(x, y)))$
- A general form of (PRP) known as unnested multiple recursion (UMR):
  (UMR) $f(x+1, y+1) = h(x, y, f(x, p(x, y)), f(x+1, y))$
- Simple nested recursion (SNR):
  (SNR) $f(x+1, y) = h(x, y, f(x, p(x, y, f(x, y))))$
- More general form (GSNR) of (SNR) with more than one recursion arguments:
  $f(x+1, y+1, z) = h(x, y, z, f(x, p(x, y, z), f(x+1, y, z)))$

Introduction 2/2

In predicative formulation (or tiered formulation) these recursion schemata make some difference.

- The class of poly-time functions can be captured with predicative (primitive) recursion.
  (Bellantoni-Cook '92, Leivant '95)
- The class of PSPACE functions can be captured with the predicative form of (UMR). (Leivant-Marion, '95)
- The class of EXPTIME functions can be captured with the predicative form of (GSNR). (Arai-E. '09)
- Observation: The predicative form of (SNR) is sound for ETIME, i.e. $2^{O(n)}$-time functions of exponential growth rates.

Outline 1/2

- To assess complexity of a given function, it is natural to look at the maximal length of rewriting sequences - runtime complexity - in the corresponding rewrite system.
  - Runtime complexity can be exponentially related to complexity of the given function. (Cichon-Weiermann '97)
  - (Innermost) runtime complexity can be polynomially related to complexity of the given function. (A.-Moser '10)
- Template: For a time-complexity class $\mathcal{F}$ of functions including polytime, find a termination order $>$ such that
  1. Every function in $\mathcal{F}$ can be represented by a rewrite system orientable with $>$. (Completeness)
  2. Runtime complexity of every rewrite system orientable with $>$ lies in $\mathcal{F}$. (Soundness)

Outline 2/2

Term-rewriting, path-ordering, characterisations of complexity classes corresponding to recursion-theoretic ones.

- The class of poly-time functions can be captured with polynomial path order (POP*). (A.-Moser '08)
- The class of PSPACE functions can be captured with light lexicographic path order (LLPO). (Cichon-Marion, unpublished)
- The class of EXPTIME functions can be captured with exponential path order EPO* (A.-E.-Moser '11).
- This talk: The class of ETIME functions can be captured with path order for ETIME (POE*).

Predicative recursion

Example (Addition)

$$\begin{aligned} add(0, y) &\to y \\ add(s(x), y) &\to s(add(x, y)) \end{aligned}$$

$add(x, y)$ computes the standard addition $x + y$.
In this example a specific argument separation is possible:

$$add(0; y) \to y \qquad add(s(x); y) \to s(; add(x; y))$$

- Called predicative recursion. (Bellantoni & Cook '92)
- Intuition: f(recursion performed; recursion term substituted):
$$\begin{cases} f(0, \vec{y}; \vec{z}) = g(\vec{y}; \vec{z}) \\ f(s(x), \vec{y}; \vec{z}) = h(x, \vec{y}; \vec{z}, f(x, \vec{y}; \vec{z})) \end{cases}$$
- Aim: to weaken the power of primitive recursion.

Polynomial path order

- Polytime functions can be characterised with predicative recursion. (Bellantoni-Cook '92)
- Generalisation of predicative recursion with polynomial path order (POP*). (A.-Moser '08)
  $f(s(x), \vec{y}; \vec{z}) >_{\mathsf{pop}*} h(x, \vec{y}; \vec{z}, f(x, \vec{y}; \vec{z}))$

Theorem (A.-Moser '08)

1. *Every polytime function can be represented by a rewrite system orientable with POP*.* *(Completeness)*
2. *The length of every (innermost) rewriting sequence (starting with an argument-normalized term) in a rewrite system orientable with POP* can be bounded by a polynomial in the size of the starting term.* *(Soundness)*

Path order for ETIME (1/2)

Introducing Path Order for ETIME (POE*) $>_{\mathsf{poe}^*}$.

- A path order is a binary relation over terms induced by a precedence.
  - A precedence is a well-founded binary relation on a signature.
  - Intuitively $f > h$ means f is defined using h.
- Mostly a path order includes the sub-term relation $\trianglerighteq$.
- E.g. recursive path orders, Knuth-Bendix order, etc.

Definition (An auxiliary relation $\rhd^n$)

$f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l}) \rhd^n t$ if $s_i \trianglerighteq^n t$ for some $i \in \{1, \ldots, k\}$.

Path order for ETIME (2/2)

Assume a precedence $>$ on an underlying signature.

Definition (Path order for ETIME)

$s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l}) >_{\mathsf{poe}^*} t$ if one of 1–3 holds.

1. $s_i \geqslant_{\mathsf{poe}*} t$ for some $i \in \{1, \ldots, k + l\}$.
2. $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$,
   - $f > g$,
   - $s \rhd^n t_j$ for all $j \in \{1, \ldots, m\}$, and
   - $s >_{\mathsf{poe}^*} t_j$ for all $j \in \{m + 1, \ldots, m + n\}$.
3. $t = f(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$,
   - $s_j \geqslant_{\mathsf{poe}*} t_j$ for all $j \in \{1, \ldots, k\}$,
   - $s_i >_{\mathsf{poe}^*} t_i$ for some $i \in \{1, \ldots, k\}$, and
   - $s >_{\mathsf{poe}^*} t_j$ for all $j \in \{k + 1, \ldots, k + l\}$.

Examples (1/4)

Example (Addition)

$$\begin{aligned} add(0; y) &\to y \\ add(s(; x); y) &\to s(; add(x; y)) \end{aligned}$$

$add(x, y)$ computes the standard addition $x + y$.
Let $add > s$. Orientation of the second rule:

1. $s(; x) >_{\mathsf{poe}^*} x$ and $add(s(; x); y) >_{\mathsf{poe}^*} y$.
2. $add(s(; x); y) >_{\mathsf{poe}^*} add(x; y)$. (by 1)
3. $add(s(; x); y) >_{\mathsf{poe}^*} s(; add(x; y))$. (by $f > s$ and 2)

Example (Multiplication)

$$\begin{aligned} mul(0, y;) &\to 0 \\ mul(s(; x), y;) &\to add(y; mul(x, y;)) \end{aligned}$$

Orientation is possible in the same way.

Examples (2/4)

Example (Exponential)

$$\begin{aligned} exp(0; y) &\to s(; y) \\ exp(s(; x); y) &\to exp(x; exp(x; y)) \end{aligned}$$

$exp(x, y)$ computes $2^x + y$.
Orientation of the second rule:

1. $s(; x) >_{\mathrm{poe}^*} x$ and $exp(s(; x); y) >_{\mathrm{poe}^*} y$.
2. $exp(s(; x); y) >_{\mathrm{poe}^*} exp(x; y)$. (by 1)
3. $exp(s(; x); y) >_{\mathrm{poe}^*} exp(x; exp(x; y))$.
(by $s(; x) >_{\mathrm{poe}^*} x$ and 2)

Examples (3/4)

Example (Linear exponential)

$$\begin{aligned} exp^1(0, y; z) &\to exp(y; z) \\ exp^1(x, 0; z) &\to exp(x; z) \\ exp^1(s(; x), s(; y); z) &\to exp^1(x, s(; y); exp^1(x, s(; y); z)) \end{aligned}$$

$exp^1(x, y, z)$ computes $2^{x+y} + z$.
Orientation of the third rule:

1. $s(; x) >_{\mathsf{poe}^*} x$ and $s(; y) \geqslant_{\mathsf{poe}*} s(; y)$.
2. $exp^1(s(; x), s(; y); z) >_{\mathsf{poe}^*} z$.
3. $exp^1(s(; x), s(; y); z) >_{\mathsf{poe}^*} exp^1(x, s(; y); z)$. (by 1 & 2)
4. $exp^1(s(; x), s(; y); z) >_{\mathsf{poe}^*} exp^1(x, s(; y); exp^1(x, s(; y); z))$. (by 1 & 3)

Examples (4/4)

Example (Quadratic exponential)

$$
\begin{aligned}
exp^2(0, y, z; w) &\rightarrow exp(z; w)\\
exp^2(x, 0, z; w) &\rightarrow exp(z; w)\\
exp^2(x, y, s(; z); w) &\rightarrow exp^2(x, y, z; exp^2(x, y, z; w))\\
exp^2(s(; x), s(; y), 0; w) &\rightarrow exp^2(s(; x), y, s(; x); w)
\end{aligned}
$$

$exp^2(x, y, z, w)$ computes $2^{x \cdot y + z} + w$.

- Orientation of the fourth rule is not possible.
- Because element-wise comparison of $(s(; x), s(; y), 0)$ and $(s(; x), y, s(; x))$ fails.

Complexity result

Theorem (Main result)

POE is sound and complete for ETIME functions,* $2^{O(n)}$*-time computable exponential functions, in the same sense as POP*.*

Note:

- The class of ETIME functions is less common than the class ETIME of predicates.
- POE* is strictly intermediate between small polynomial path order (sPOP*) and exponential path order (EPO*).
  1. sPOP* is sound and complete for polytime functions. (A.-E.-Moser '12)
  2. EPO* is sound and complete for EXPTIME functions. (A.-E.-Moser '11)
- All of sPOP*, POE* and EPO* are weak sub-relations of recursive path orders.

Contrast to related path orders

The difference among sPOP*, POE* and EPO* lies only in case of recursive comparison $f(\cdots;\cdots) >_{\mathsf{poe}^*} f(\cdots;\cdots)$:

	Complexity	Comparison
sPOP*	$n^{O(1)}$	f(element-wise ; element-wise)
POE*	$2^{O(n)}$	f(element-wise ; full comparison)
EPO*	$2^{n^{O(1)}}$	f(lexicographic ; full comparison)

	sPOP*	POE*	EPO*
add, *mul*	✓	✓	✓
exp, exp^1	—	✓	✓
exp^2	—	—	✓

(—: not orientable, ✓: orientable)

Summary

- The class of primitive recursive functions is closed under several non-trivial recursion schemata.
- But those recursion schemata might make a difference for smaller classes - complexity classes - in the predicative setting.
  - Predicative primitive recursion corresponds to polytime functions.
  - Predicative simple nested recursion corresponds to ETIME functions.
  - Predicative simple nested recursion with more than one recursion arguments corresponds to EXPTIME functions.
- Path orders sPOP*, POE* and EPO* essentially encodes these predicative recursion schemata.

Conclusion

- Based on a close connection between time-complexity and runtime complexity, a new path order POE* is introduced characterising ETIME computable functions.
- Asking whether there is a uniform machinery to characterise complexity classes independent of machine models.
- Further question: Is there is a uniform soundness proof e.g. for sPOP*, POE* and EPO*?

Thank you for your attention!

Speaker is supported by JSPS postdoctoral fellowships for young scientists.