Handout for part 5:

Complexity:
tuple interpretations

1. Monotonic algebras

2

Derivation height

A measure of the "cost" of reducing a term to normal form (worst-case).

$$\begin{array}{rcl} \mathtt{add}(x,\mathtt{0}) & \to & x \\ \mathtt{add}(x,\mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{add}(x,y)) \\ \mathtt{mul}(x,\mathtt{0}) & \to & \mathtt{0} \\ \mathtt{mul}(x,\mathtt{s}(y)) & \to & \mathtt{add}(x,\mathtt{mul}(x,y)) \end{array}$$

Derivation height:

- $\mathtt{add}(\mathtt{0},\mathtt{s}(\mathtt{0}))$: 2 ($\mathtt{add}(\mathtt{0},\mathtt{s}(\mathtt{0})) \to \mathtt{s}(\mathtt{add}(\mathtt{0},\mathtt{0})) \to \mathtt{s}(\mathtt{0})$).
- $\mathtt{mul}(\ \mathtt{mul}(\mathtt{s}(\mathtt{s}(\mathtt{0})),\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))\ ,\ \mathtt{0})$: 15

3

Traditional interpretations (first-order)

Idea:

- map every term s to $[\![s]\!] \in \mathbb{N}$
- make sure that $s \to t$ implies $[\![s]\!] > [\![t]\!]$

Then: $[\![s]\!] \geq \mathtt{derivationheight}(s)$!

Approach:

- map every function that takes k arguments to a **monotonic** function in $\mathbb{N}^k \mapsto \mathbb{N}$
- make sure that $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$

4

Bounding derivation height with interpretations to $\mathbb{N}$

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

Let:

- $[\![\mathtt{0}]\!] = 0$
- $[\![\mathtt{s}(x)]\!] = x + 1$
- $[\![\mathtt{add}(x,y)]\!] = 1 + y + 2 * x$

We might initially be inclined to choose $[\![\texttt{add}(x,y)]\!] = x+y$ – but then we do not have that $[\![\ell]\!] > [\![r]\!]$ for the rules. Hence, the interpretation cannot exactly match the "meaning" of the rules:

Then:

$$\begin{array}{rcccl} [\![\texttt{add}(\texttt{0},y)]\!] & = & 1+y & > & [\![y]\!] \\ [\![\texttt{add}(\texttt{s}(x),y)]\!] & = & 3+y+2*x & > & 2+y+2*x \\ & & & = & [\![\texttt{s}(\texttt{add}(x,y))]\!] \end{array}$$

Hence: $[\![\texttt{add}(\texttt{s}^n(\texttt{0}),\ \texttt{s}^m(\texttt{0}))]\!] = 1+m+2*n$: linear!

5

Monotonic algebras: definition

Given: a set $\mathcal{A}$ with a well-founded ordering $>$ (for example: $\mathbb{N}$)

Choose: a function $[\texttt{f}]$ from $\mathcal{A}^k$ to $\mathcal{A}$ for every $\texttt{f}$ of arity k

Define: for a given α mapping variables to $\mathcal{A}$:

- $[\![x]\!] = \alpha(x)$
- $[\![\texttt{f}(s_1,\dots,s_k)]\!] = [\texttt{f}]([\![s_1]\!],\dots,[\![s_k]\!])$

Prove: $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$, all α

In practice, since we quantify over α, we essentially view both sides as functions over a given set of variables. This is why we for instance write $[\![\texttt{add}(\texttt{0},y)]\!] = 1+y$ instead of $1+\alpha(y)$.

Then: $[\![s]\!] > [\![t]\!]$ whenever $s \to_{\mathcal{R}} t$.

The most common example is to choose the set of natural numbers for $\mathcal{A}$, but we could also for instance choose the rational numbers (with $x > y$ if $x \geq y+1$), or pairs of numbers as we will see later.

Consequence: if $\mathit{tonat}(a) > \mathit{tonat}(b)$ whenever $a > b$ then $\mathit{tonat}([\![s]\!]) \geq \texttt{derivationheight}(s)$. (Here, we let *tonat* be a function that maps each element of $\mathcal{A}$ to a natural number. If $\mathcal{A} = \mathbb{N}$ this is just the identity; if $\mathcal{A} = \mathbb{Q}$ this could for instance be rounding down.)

6

Higher-order interpretations to $\mathbb{N}$: problems

Let's extend this idea to higher-order rewriting. Here, we quickly run into the problem: what to do with partial applications? For example:
Suppose: $[\![\texttt{s}(x)]\!] = x+1$
Question: What is $[\![\texttt{s}]\!]$?

Problem: behaviour matters!

$$\begin{array}{rcl} \texttt{fold}(F,x,[]) & \to & [] \\ \texttt{fold}(F,x,\texttt{cons}(y,l)) & \to & \texttt{fold}(F,(F\cdot x\cdot y),l) \\ \texttt{add}(x,\texttt{0}) & \to & x \\ \texttt{add}(x,\texttt{s}(y)) & \to & \texttt{s}(\texttt{add}(x,y)) \end{array}$$

- What is the derivation height if $F := \lambda x,y.\texttt{minimum}(x,y)$?
- What if: $F := \lambda x,y.\texttt{add}(x,y)$?

- What if: $F := \lambda x, y.\mathtt{add}(x, \mathtt{s}(\mathtt{0}))$?
- What if: $F := \lambda x, y.\mathtt{add}(x, \mathtt{s}(\mathtt{s}(\mathtt{0})))$?
- What if: $F := \lambda x, y.\mathtt{add}(x, x)$?

All in all, the consequences of using different functions for F cannot really be captured by a number.

7

Proposal

Let's interpret terms of function type as functions!

More than that: for each type we have a possibly different interpretation domain. We only fix that function types are interpreted as *monotonic* functions:

Type interpretations:

- For every **base type** ι: a set $\mathcal{A}_\iota$, ordering $>_\iota$ and quasi-ordering $\geq_\iota$
- Define:

$$\begin{array}{rcl} (\!|\iota|\!) & = & \mathcal{A}_\iota \\ (\!|\sigma \Rightarrow \tau|\!) & = & \text{“monotonic functions from } (\!|\sigma|\!) \text{ to } (\!|\tau|\!)\text{”} \\ F >_{\sigma\Rightarrow\tau} G & \text{if} & F(a) >_\tau G(a) \text{ for all } a \in (\!|\sigma|\!) \\ F \geq_{\sigma\Rightarrow\tau} G & \text{if} & F(a) \geq_\tau G(a) \text{ for all } a \in (\!|\sigma|\!) \end{array}$$

8

Higher-order monotonic algebras: definition

(Difference to the first-order definition are indicated in red.)

Given: a a *type interpretation function* as on the previous slide

Choose: a function $[\mathtt{f}]$ in $(\!|\sigma|\!)$ for every $\mathtt{f}$ of type σ

Define: for a given α mapping variables to $\mathcal{A}$:

- $[\![x]\!] = \alpha(x)$
- $[\![\mathtt{f}]\!] = [\mathtt{f}]$
- $[\![s \cdot t]\!] = [\![s]\!]([\![t]\!])$

(We're ignoring abstractions for now. We will get back to that later!)

Prove: $[\![\ell]\!] > [\![r]\!]$ for all rules $\ell \to r$, all α

In practice, since we quantify over α, we essentially view both sides as functions over a given set of variables.

Then: $[\![s]\!] > [\![t]\!]$ whenver $s \to_{\mathcal{R}} t$.

Consequence: if $\mathit{tonat}(a) > \mathit{tonat}(b)$ whenever $a > b$ then $\mathit{tonat}([\![s]\!]) \geq \mathtt{derivationheight}(s)$.

Note that of course, this is also a *termination* technique: if we have a bound on the number of steps, clearly this number is not infinite.

9
Example:

$$\begin{array}{rcl} [] & :: & \mathsf{list} \\ \mathtt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \mathtt{map} & :: & (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Choose: $\mathcal{A}_\iota = \mathbb{N}$ for all ι

$$\begin{array}{rcl} [[]] & = & 0 \\ [\mathtt{cons}](x, y) & = & x + y + 1 \\ [\mathtt{map}](F, x) & = & (x + 1) * F(x) \end{array}$$

Monotonicity: holds. (We can easily see that, for example, if $x > y$ then $[\mathtt{map}](F, x) > [\mathtt{map}](F, y)$, and if $F(x) > G(x)$ for all x then $[\mathtt{map}](F, x) > [\mathtt{map}](G, x)$.)

10
Example

$$\begin{array}{rcl} [[]] & = & 0 \\ [\mathtt{cons}](x, y) & = & x + y + 1 \\ [\mathtt{map}](F, x) & = & (x + 1) * F(x) + 1 \end{array}$$

Goal 1:

$$[\![\mathtt{map}(F, [])]\!] > [\![[]]\!]$$

That is:

$$(0 + 1) * F(0) + 1 > 0$$

Which is certainly true because $1 > 0$.

Goal 2:

$$[\![\mathtt{map}(F, \mathtt{cons}(x, l))]\!] > [\![\mathtt{cons}(F \cdot x, \mathtt{map}(F, l))]\!]$$

That is:

$$\begin{array}{l} ((x + l + 1) + 1) * F(x + l + 1) + 1 \quad > \\ F(x) + ((l + 1) * F(l) + 1) + 1 \end{array}$$

Simplifying the arithmetic, this is:

$$\begin{array}{l} x * F(x + l + 1) + l * F(x + l + 1) + F(x + l + 1) + F(x + l + 1) + 1 \quad > \\ F(x) + l * F(l) + F(l) + 1 \end{array}$$

Let's reorganise that a bit!

$$\begin{array}{lllll} x * F(x + l + 1) & + l * F(x + l + 1) & + F(x + l + 1) & + F(x + l + 1) & +1 \\ > & + l * F(l) & + F(x) & + F(l) & +1 \end{array}$$

Now observe that F is *monotonic*. So for instance $F(x+l+1) > F(x)$. Hence we quickly see that this inequality indeed holds.

11

Exercise

Given:

$$\begin{array}{rcl} [] & :: & \mathsf{list} \\ \mathtt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \mathtt{filter} & :: & (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \mathtt{helper} & :: & \mathsf{bool} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \mathtt{filter}(F, []) & \to & [] \\ \mathtt{filter}(F, \mathtt{cons}(x, l)) & \to & \mathtt{helper}(F \cdot x, x, \mathtt{filter}(F, l)) \\ \mathtt{helper}(\mathtt{true}, x, l) & \to & \mathtt{cons}(x, l) \\ \mathtt{helper}(\mathtt{false}, x, l) & \to & l \end{array}$$

Task: show that the following interpretation suffices:

$$\begin{array}{rcl} [[]] & = & 0 \\ [\mathtt{cons}](x, y) & = & x + y + 1 \\ [\mathtt{helper}](b, x, y) & = & b + x + y + 1 \\ [\mathtt{filter}](F, x) & = & (x+1) * (F(x) + 1) \end{array} \qquad \begin{array}{rcl} [\mathtt{true}] & = & 1 \\ [\mathtt{false}] & = & 0 \end{array}$$

12

Bonus exercise

Given:

$$\begin{array}{rcl} [] & :: & \mathsf{list} \\ \mathtt{cons} & :: & \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \mathtt{zip} & :: & (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \mathtt{zip}(F, [], l) & = & l \\ \mathtt{zip}(F, l, []) & = & l \\ \mathtt{zip}(F, \mathtt{cons}(x, l), \mathtt{cons}(y, q)) & = & \mathtt{cons}(F \cdot x \cdot y, \mathtt{zip}(F, l, q)) \end{array}$$

Task: find an interpretation that orients these rules!

13

Abstraction

Discussion: what should be the interpretation of $\lambda x.s$?

Naive choice: $x \mapsto [\![s]\!]$

Problem: the naive interpretation for for $\lambda x.s$ is not monotonic if x does not occur in s! For example, this choice would let $[\![\lambda x.0]\!]$ be the **constant** function mapping everything to 0 – and thus, it would not be an element of $(\!|\mathsf{nat} \Rightarrow \mathsf{nat}|\!)$.

Solution: for each σ, τ, a function $\mathtt{makesm}_{\sigma,\tau}$:

- Input: a monotonic **or** constant function from $(\!|\sigma|\!)$ to $(\!|\tau|\!)$
- Output: a monotonic function from $(\!|\sigma|\!)$ to $(\!|\tau|\!)$

- $\mathtt{makesm}_{\sigma,\tau}$ should itself be monotonic!
- we need to have $[\![(\lambda x.s)\cdot t]\!] > s[x := t]$

The use of `makesm` functions may be confusing at first – but essentially, all that this means is that we choose a systematic way of turning a given abstraction into a monotonic function. And in practice, we can usually find a way to define a class of `makesm` functions that allows us to *almost* map $\lambda x.s$ to $x \mapsto [\![s]\!]$ if $x \in FV(s)$ – just adding a cost for the β-reduction. This is demonstrated for $\mathcal{A}_{\mathsf{nat}} = \mathbb{N}$ below.

Example: (for $\sigma, \tau = \mathsf{nat}$ and $\mathcal{A}_{\mathsf{nat}} = \mathbb{N}$):

- if F is constant, then $\mathtt{makesm}_{\sigma,\tau}(F) = x \mapsto F(x) + x + 1$
- otherwise $\mathtt{makesm}_{\sigma,\tau}(F) = x \mapsto F(x) + 1$

This definition works very nicely in practice. The only difficulty is to prove that the above `makesm` function is indeed monotonic; in particular, if F is monotonic in x and G is constant, we must show that that $F >_{\mathsf{nat}\Rightarrow\mathsf{nat}} G$ implies that also $\mathtt{makesm}(F) >_{\mathsf{nat}\Rightarrow\mathsf{nat}} \mathtt{makesm}(G)$. To see that this holds, we make the observation that in the natural numbers, if F is a monotonic function, then $F(x+1) > F(x)$, so $F(x+1) \geq F(x)+1$; by induction, we see that $F(n) \geq F(0)+n$. In a constant function, $G(n) = G(0)$. Thus we see: for all n: $F(n) \geq F(0) + n > G(0) + n = G(n) + n$.

This idea can be generalised to all types, but it takes a bit more definition effort; for example, if $\sigma = \tau = \mathsf{nat} \Rightarrow \mathsf{nat}$ we let $\mathtt{makesm}_{\sigma,\tau}(F) = (G,x) \mapsto F(G,x)+1$ if F is monotonic in its first argument (G), and $(G,x) \mapsto F(G,x) + G(0) + 1$ if F is constant in its first argument.

2. Tuple interpretations

14
An observation

Consider:

- $[\![\mathtt{add}(\mathtt{s}^n(\mathtt{0}),\ \mathtt{s}^m(\mathtt{0}))]\!] = 1 + m + 2 * n$
- actual cost of reduction: $n + 1$
- size of normal form: $n + m$
- This does raise the question: are we actually giving a bound to the *sum* of cost and size by using interpretations to $\mathbb{N}$?

Idea: separate cost and size already in the interpretation!

Mechanism: map to $\mathbb{N}^2$ instead of $\mathbb{N}$.

We let $\langle x, y\rangle > \langle x', y'\rangle$ if $x > x'$ and $y \geq y'$.

Note: we can choose $tonat(\langle x, y\rangle) = x$. That is, if $a > b$ in $\mathbb{N}^2$ then $tonat(a) > tonat(b)$ – so if we can express $[\![s]\!]$ as an element $\langle x, y\rangle$ of $\mathbb{N}^2$, then x gives a bound on the derivation height of s. We will refer to the first element of the tuple as the **cost component** of the tuple.

15
Separating cost and size

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

Let:

$$\begin{array}{lcccccc} & & \textbf{cost} & & \textbf{size} & \\ [\![\mathtt{0}]\!] & = \langle & 0 & , & 0 & \rangle \\ [\![\mathtt{s}(x)]\!] & = \langle & x_{cost} & , & x_{size} + 1 & \rangle \\ [\![\mathtt{add}(x, y)]\!] & = \langle & x_{cost} + y_{cost} + x_{size} & , & x_{size} + y_{size} & \rangle \end{array}$$

Then:

$$\begin{array}{rclcl} [\![\mathtt{add}(\mathtt{0}, y)]\!] & = & \langle 1 + y_1, y_2\rangle & & \\ & > & \langle y_1, y_2\rangle & = & [\![y]\!] \\ [\![\mathtt{add}(\mathtt{s}(x), y)]\!] & = & \langle 2 + x_1 + y_1 + x_2, 1 + x_2 + y_2\rangle & & \\ & > & \langle 1 + x_1 + y_1 + x_2, 1 + x_2 + y_2\rangle & = & [\![\mathtt{s}(\mathtt{add}(x, y))]\!] \end{array}$$

Hence: $[\![\mathtt{add}(\mathtt{s}^n(\mathtt{0}),\ \mathtt{s}^m(\mathtt{0}))]\!] = \langle 1 + n, n + m\rangle$: precise! (And also intuitive.)

16
When interpretations to $\mathbb{N}$ are Not Great

$$\mathtt{a}(\mathtt{b}(x)) \to \mathtt{b}(\mathtt{a}(x))$$

Let:

- $[\![\mathtt{a}(x)]\!] = 2 * x$
- $[\![\mathtt{b}(x)]\!] = x + 1$
- $[\![\epsilon]\!] = 0$

Then:

$$[\![\mathtt{a}(\mathtt{b}(x))]\!] \quad = \quad 2 + 2 * x > 1 + 2 * x \quad = \quad [\![\mathtt{b}(\mathtt{a}(x))]\!]$$

Hence: $[\![\mathtt{a}^n(\mathtt{b}^m(\epsilon))]\!] = 2^n * m$: exponential!

17

Separating cost and size

$$\mathtt{a}(\mathtt{b}(x)) \quad \to \quad \mathtt{b}(\mathtt{a}(x))$$

Let:

$$\begin{array}{llccccc} & & & \textbf{cost} & & \textbf{size} & \\ [\![\mathtt{a}(x)]\!] & = \langle & & x_{cost} + x_{size} & , & x_{size} & \rangle \\ [\![\mathtt{b}(x)]\!] & = \langle & & x_{cost} & , & x_{size} + 1 & \rangle \\ [\![\epsilon]\!] & = \langle & & 0 & , & 0 & \rangle \end{array}$$

Then:

$$[\![\mathtt{a}(\mathtt{b}(x))]\!] \quad = \quad \langle x_1 + x_2 + 1, x_2 + 1\rangle > \langle x_1 + x_2, x_2 + 1\rangle \quad = \quad [\![\mathtt{b}(\mathtt{a}(x))]\!]$$

Hence: $[\![\mathtt{a}^n(\mathtt{b}^m(\epsilon))]\!] = (n * m, m)$: precise!

Of course, we can't always get precision. But we invariably get tighter interpretations by using tuples than single numbers.

18

Tuple interpretations

Definition: monotonic algebras with $\mathcal{A}_\iota = \mathbb{N}^{K[\iota]}$ for all ι (where $K[\iota]$ is a positive integer for all ι).

$\Longrightarrow$ both for first- and higher-order!

This is a specific implementation of a well-known method (monotonic algebras), that adds a surprising amount of power over other variations. In the bigger picture, tuple interpretations can be seen as a generalisation of the method of *matrix interpretations*: this method also considers tuples over $\mathbb{N}$ as the interpretation domain, but restrict the shape of the interpretation functions $[\mathtt{f}]$.

Of course, there is no reason to stop here. We could have tuples over *other* sets than $\mathbb{N}$ – for example, using the set of integers $\mathbb{Z}$ as the second set in the component (as only the first needs to admit a wellfounded ordering), a set such as $\mathbb{N} \cup \{\infty\}$, or even some impromptu set $\{a, b, c\}$ with $a > b$ and $a > c$ but b, c not comparable. There are uses for all these examples. We could also use tuples only for *some* base types, and still allow, for instance, a base type $\mathsf{list}(\mathbf{N} \Rightarrow \mathbf{N})$ to be mapped to a function space such as $(\!|\mathbf{N} \Rightarrow \mathbf{N}|\!)$. However, for this lecture, we will limit interest to tuples of the form $\mathbb{N}^k$.

Example sort interpretations:

- $\{\mathsf{nat}\} = \mathbb{N}^2$ (cost, size of normal form)
- $\{\mathsf{list}\} = \mathbb{N}^3$ (cost, list length, size of greatest element)

- $\{\mathsf{bool}\} = \mathbb{N}^1$ (cost)

19

Example: interpreting list functions

$$\begin{array}{rcl} \mathtt{append}([], l) & \to & l \\ \mathtt{append}(\mathtt{cons}(x, l), q) & \to & \mathtt{cons}(x, \mathtt{append}(l, q)) \\ \mathtt{sum}([]) & \to & \mathtt{0} \\ \mathtt{sum}(\mathtt{cons}(x, l)) & \to & \mathtt{add}(x, \mathtt{sum}(l)) \end{array}$$

Interpretations:

- $\{\mathsf{list}\} = \mathbb{N}^3$ (cost, list length, maximum element)
- $[\![[]]\!] = \langle 0, 0, 0 \rangle$
- $[\![\mathtt{cons}(x, l)]\!] = \langle x_{cost} + l_{cost}, l_{len} + 1, \max(x_{size}, l_{max}) \rangle$
- $[\![\mathtt{append}(l, q)]\!] = \langle \text{cost}, \text{length}, \text{maximum} \rangle$, where:
  - maximum $= \max(l_{max}, q_{max})$
  - length $= l_{len} + q_{len}$
  - cost $= l_{cost} + q_{cost} + l_{len} + 1$
- $[\![\mathtt{sum}(l)]\!] = \langle \text{cost}, \text{size} \rangle$, where:
  - size $= l_{len} * l_{max}$
  - cost $= l_{cost} + 2 * l_{len} + l_{len} * l_{max} + 1$

20

Higher-order tuple interpretations: an example

$$\begin{array}{rcl} [] & :: & \mathsf{list} \\ \mathtt{cons} & :: & \mathbf{N} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \\ \mathtt{map} & :: & (\mathbf{N} \Rightarrow \mathbf{N}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list} \end{array}$$

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Let:

- $[\![[]]\!] = \langle 0, 0, 0 \rangle$
- $[\![\mathtt{cons}(x, l)]\!] = \langle x_{cost} + l_{cost}, l_{len} + 1, \max(x_{size}, l_{max}) \rangle$
- $[\![\mathtt{map}(F, l)]\!] = \langle \text{cost}, \text{length}, \text{maximum} \rangle$, where:
  - length: l_{len}
  - maximum: $F(\ \langle l_{cost}, l_{max} \rangle\ )_s$
  - cost: $(l_{len} + 1) * (F(\ \langle l_{cost}, l_{max} \rangle\ )_{cost} + 1)$

21

Exercise

1. Find an interpretation, with $(\!|\mathsf{nat}|\!) = \mathbb{N}^2$, for the following system:

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

   Warning: do not take $x_{size} - y_{size}$ for the size of $\mathtt{minus}(x, y)$! Doing this would break the monotonicity requirement: we must have $[\![\mathtt{minus}(a, b)]\!] > [\![\mathtt{minus}(a, c)]\!]$ if $b > c$, which implies $[\![\mathtt{minus}(a, b)]\!]_{size} \geq [\![\mathtt{minus}(a, c)]\!]_{\geq}$ if $b_{cost} > c_{cost}$ and $b_{size} \geq c_{size}$.

   Side note: the fact that we can do this at all illustrates the power of tuple interpretations. This was a motivating example for dependency pairs, since it cannot be handled with any well-founded ordering that has $\mathtt{minus}(x, y) \succeq y$. Thus, termination *cannot* be proved using RPO or interpretations to $\mathbb{N}$, nor can it be proved with a method like matrix interpretations due to the duplication of x in the last rule. Yet, here we do not only prove its termination, but also find a bound to its complexity.

2. Find an interpretation for the following HTRS, where $\mathtt{zip} :: (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$.

$$\begin{array}{rcl} \mathtt{zip}(F, [], l) & = & l \\ \mathtt{zip}(F, l, []) & = & l \\ \mathtt{zip}(F, \mathtt{cons}(x, l), \mathtt{cons}(y, q)) & = & \mathtt{cons}(F \cdot x \cdot y, \mathtt{zip}(F, l, q)) \end{array}$$

22

A more challenging higher-order tuple interpretation

$$\begin{array}{rcl} \mathtt{fold}(F, x, []) & \to & [] \\ \mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, (F \cdot x \cdot y), l) \end{array}$$

Interpretation:

$$[\![\mathtt{fold}(F, x, l)]\!] \quad = \quad \langle \text{cost}, \ \text{size} \rangle$$

Where:

- $\text{cost} = 1 + l_{cost} + F(\langle 0, 0 \rangle)_{cost} + \mathit{Helper}[F, \langle l_{cost}, l_{max} \rangle]^{l_{len}}(x)_{cost}$
- $\text{size} = \mathit{Helper}[F, \langle l_{cost}, l_{max} \rangle]^{l_{len}}(x)_{size}$
- And $\mathit{Helper}[F, y] = x \mapsto \langle F(x, y)_{cost}, \max(x_{size}, F(x, y)_{size}) \rangle$.

23

A more challenging higher-order tuple interpretation

$$\begin{array}{rcl} \mathtt{add}(\mathtt{0}, y) & \to & y \\ \mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\ \mathtt{fold}(F, x, []) & \to & [] \\ \mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, (F \cdot x \cdot y), l) \\ \mathtt{sum}(l) & \to & \mathtt{fold}(\lambda x.\lambda y.\mathtt{add}(x, y), \mathtt{0}, l) \end{array}$$

Method: Plug $[\![\lambda x.\lambda y.\mathtt{add}(x, y)]\!]$ into the interpretation for `fold`.

Interpreting λ: use $\mathtt{makesm}_{\iota,\sigma_1\Rightarrow\ldots\Rightarrow\sigma_m\Rightarrow\kappa} =$

$$\begin{cases} (F, x, y_1, \ldots, y_m) & \mapsto & (F(x, \vec{y})_1 + 1 + x_1, F(x, \vec{y})_2, \ldots, F(x, \vec{y})_{K[\kappa]}) \text{ if } F \text{ is constant} \\ (F, x, y_1, \ldots, y_m) & \mapsto & (F(x, \vec{y})_1 + 1 \qquad\quad , F(x, \vec{y})_2, \ldots, F(x, \vec{y})_{K[\kappa]}) \text{ if } F \text{ is monotonic} \end{cases}$$

3. Complexity notions

24

Derivational and runtime complexity (first-order)

Derivational complexity:

$n \mapsto$ "maximum derivation height for a term of size n"

Downside: can easily get large; e.g.: $\mathtt{mul}(\mathtt{mul}(\mathtt{mul}(\mathtt{mul}(\mathtt{s}(\mathtt{s}(0)), \mathtt{s}(\mathtt{s}(0))), \mathtt{s}(\mathtt{s}(0))), \mathtt{s}(\mathtt{s}(0))), \mathtt{s}(\mathtt{s}(0)))$

Runtime complexity:

$n \mapsto$ "maximum derivation height for a **basic** term of size n"

Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$

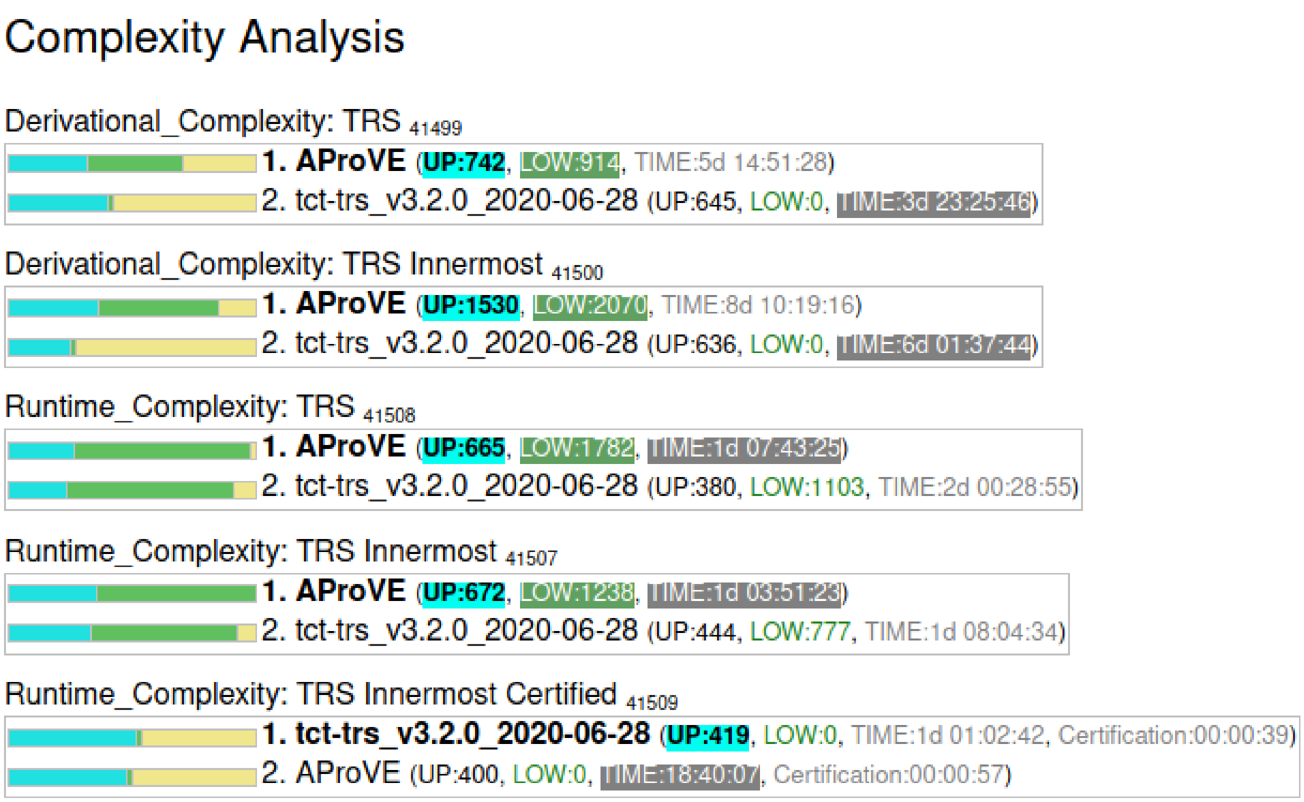
Example: $\mathtt{mul}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0))))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0))))))))$

Connection with computational complexity: depends

25

Termination (and complexity) competition

In the annual termination competition, there are categories for both runtime and derivational complexity of first-order term rewriting (both with a general reduction strategy, and focused on innermost reduction).



26

Complexity of higher-order term rewriting

Open question: do derivational and runtime complexity even make sense for higher-order rewriting?

$$\begin{array}{rcl} \mathtt{fold}(F, x, []) & \to & [] \\ \mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, (F \cdot x \cdot y), l) \end{array}$$

Recall:

- What if: $F := \lambda x, y.\mathtt{minimum}(x, y)$?
- What if: $F := \lambda x, y.\mathtt{add}(x, y)$?
- What if: $F := \lambda x, y.\mathtt{add}(x, x)$?

27

Higher-order derivational complexity?

Idea: naively extend the definition of derivational complexity

Result:

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

- $(\lambda x.\mathtt{add}(x, x)) \cdot (\mathtt{s}(\mathtt{s}(\mathtt{0})))$
- $(\lambda x.\mathtt{add}(x, x)) \cdot ((\lambda x.\mathtt{add}(x, x)) \cdot (\mathtt{s}(\mathtt{s}(\mathtt{0}))))$
- $(\lambda x.\mathtt{add}(x, x)) \cdot ((\lambda x.\mathtt{add}(x, x)) \cdot ((\lambda x.\mathtt{add}(x, x)) \cdot (\mathtt{s}(\mathtt{s}(\mathtt{0})))))$
- ...

Conclusion: exponential complexity at a minimum, even for very simple systems.

28

Runtime complexity: a simple extension

Runtime complexity:
$n \mapsto$ "maximum derivation height for a **basic** term of size n"
Basic term: $\mathtt{function}(\mathtt{data}, \ldots, \mathtt{data})$

Question: is it interesting to look at λ-functions over constructors?

- $\mathtt{map}(\lambda x.\mathtt{s}(x), \text{some lst})$?
- $\mathtt{maketree}(\lambda x_{\mathsf{nat}}, y_{\mathsf{tree}}.\mathtt{node}(x, y, y), \text{some natural number})$

A notion of runtime complexity like this would be well-defined, and give reasonable bounds. However, where runtime complexity makes sense in first-order rewriting if we are interested in "start terms" for a program, the concept of instantiating higher-order functions by constructors or functions that are built from constructors doesn't seem to have much practical relevance.

Choice: data must be a **first-order** term.

Thus, we let the start terms for higher-order runtime complexity analysis be *exactly the same* as those for runtime analysis of first-order term rewriting. Yet, higher-order function calls may arise during the evaluation of the start terms, so their analysis is still needed. This actually seems representative of full program analysis.

29

Higher-order runtime complexity example

$$
\begin{array}{rcl}
\mathtt{add}(\mathtt{0}, y) & \to & y \\
\mathtt{add}(\mathtt{s}(x), y) & \to & \mathtt{add}(x, \mathtt{s}(y)) \\
\mathtt{fold}(F, x, []) & \to & [] \\
\mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, (F \cdot x \cdot y), l) \\
\mathtt{sum}(l) & \to & \mathtt{fold}(\lambda x.\lambda y.\mathtt{add}(x, y), \mathtt{0}, l)
\end{array}
$$

Basic terms:

- $\mathtt{add}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))), \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))))))))$
- $\mathtt{sum}(\mathtt{cons}(\mathtt{s}(\mathtt{s}(\mathtt{0})), \mathtt{cons}(\mathtt{0}, \mathtt{cons}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{0}))), []))))$

Runtime complexity: $n \mapsto \mathcal{O}(n^2)$ (actually: length * max)

30

Exercises

1. Compute a bound on the runtime complexity of the following system.

$$
\begin{array}{rcl}
\mathtt{map}(F, []) & \to & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \\
\mathtt{doublemap}(l) & \to & \mathtt{map}(\mathtt{double}, l) \\
\mathtt{double}(\mathtt{0}) & \to & \mathtt{0} \\
\mathtt{double}(\mathtt{s}(x)) & \to & \mathtt{s}(\mathtt{s}(\mathtt{double}(x)))
\end{array}
$$

2. Compute a bound on the runtime complexity of the following system.

$$
\begin{array}{rcl}
\mathtt{add}(x, \mathtt{0}) & \to & x \\
\mathtt{add}(x, \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{add}(x, y)) \\
\mathtt{zip}(F, [], l) & = & l \\
\mathtt{zip}(F, l, []) & = & l \\
\mathtt{zip}(F, \mathtt{cons}(x, l), \mathtt{cons}(y, q)) & = & \mathtt{cons}(F \cdot x \cdot y, \mathtt{zip}(F, l, q)) \\
\mathtt{zipadd}(l, q) & \to & \mathtt{zip}(\lambda x.\lambda y.\mathtt{add}(y, x), l, q)
\end{array}
$$

31

A higher-order complexity notion?

Extending the first-order runtime complexity notion to higher-order rewriting is a good start, but it doesn't really capture the higher-order nature. And indeed, tuple interpretations give us much more information, that we could use for both time and space bounds. Even just sticking to time (or: computation cost) bounds, it would be nice if we could express the complexity of functions, rather than full programs; for example:

Idea:

- complexity of `map` is $\mathcal{O}(n * F(n))$?
- complexity of `fold` is $\mathcal{O}(F^n(n))$?

However, this is speculative; there is no clear definition of what it would mean. We could likely define something, but would it be useful?

32

Basic Feasible Functions

But there *is* a higher-order version of PTIME! This is defined in terms of Turing Machines.

Idea:

- Oracle Turing Machines: these take n functions, k binary words
- to compute function i:
  - copy input to tape i
  - go to special state
  - output is written on tape $n+i$
- $\Longrightarrow$ function **cost** is assumed zero, but function **output size** is important
- Question: is the execution time limited by a **higher-order polynomial** over $F_1, \dots, F_n, w_1, \dots, w_k$?

Relevance: this is exactly determined by the existence of a higher-order polynomially-bounded tuple interpretation, provided we impose some restrictions on the interpretation of binary words.