# Uniform Resource Analysis by Rewriting

## Strengths and Weaknesses

Georg Moser

Department of Computer Science
University of Innsbruck


2nd FSCD, September 5, 2017

# Outline

# A Textbook Example

## Example

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

# A Textbook Example

## Example

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

 says

What is the complexity of `rev`?

# A Textbook Example

## Example

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

 says

What is the complexity of `rev`?

 says

Ahem, I'm so sorry, but what do you mean by "complexity"?

# A Textbook Example

## Example

```
let rec fold_left f acc = function
      [] → acc
    | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

 says

What is the complexity of `rev`?

 says

Ahem, I'm so sorry, but what do you mean by "complexity"?

 says

Sigh, runtime complexity, of course!

# Runtime Complexity, Of Course!

## Example (revisited)

$$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil}) \qquad \mathsf{rev}'(\mathsf{nil}, acc) \to acc$$
$$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$$

# Runtime Complexity, Of Course!

## Example (revisited)

$$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil}) \qquad \mathsf{rev}'(\mathsf{nil}, acc) \to acc$$
$$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$$

## Definitions

- a term rewrite system (TRS) $\mathcal{R}$ is a finite set of rules
- a computation of $\mathcal{R}$ is the application of the rules from left to right

# Runtime Complexity, Of Course!

## Example (revisited)

$$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil}) \qquad \mathsf{rev}'(\mathsf{nil}, acc) \to acc$$
$$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$$

## Definitions

- a term rewrite system (TRS) $\mathcal{R}$ is a finite set of rules
- a computation of $\mathcal{R}$ is the application of the rules from left to right

## Example

$$\begin{aligned}\mathsf{rev}(1 :: 2 :: 3 :: \mathsf{nil}) &\to_{\mathcal{R}} \mathsf{rev}'(1 :: 2 :: 3 :: \mathsf{nil}, \mathsf{nil})\\ &\to_{\mathcal{R}} \mathsf{rev}'(2 :: 3 :: \mathsf{nil}, 1 :: \mathsf{nil})\\ &\to^*_{\mathcal{R}} \mathsf{rev}'(\mathsf{nil}, 3 :: 2 :: 1 :: \mathsf{nil})\\ &\to_{\mathcal{R}} 3 :: 2 :: 1 :: \mathsf{nil}\end{aligned}$$

## Definition

a TRS $\mathcal{R}$ is terminating if $\rightarrow_{\mathcal{R}}$ is well-founded

## Definition

a TRS $\mathcal{R}$ is terminating if $\rightarrow_{\mathcal{R}}$ is well-founded

## Definitions

- a function symbol $f$ is called defined if it occurs as the root symbol on the left of a rule, otherwise $f$ is called constructor
- values are terms made up from constructors and variables

## Definition

a TRS $\mathcal{R}$ is terminating if $\to_{\mathcal{R}}$ is well-founded

## Definitions

- a function symbol $f$ is called defined if it occurs as the root symbol on the left of a rule, otherwise $f$ is called constructor
- values are terms made up from constructors and variables

## Definition

we define the runtime complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dh}_{\mathcal{R}}(t) = \max\{n \mid \exists u\ t \to_{\mathcal{R}}^{n} u\}$$

$$\mathrm{rc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n \text{ and } t \text{ is basic}\}$$

a term $f(t_1, \ldots, t_n)$ is basic if $f$ is defined and all $t_i$ are values

## Definition

a TRS $\mathcal{R}$ is terminating if $\to_{\mathcal{R}}$ is well-founded

## Definitions

- a function symbol $f$ is called defined if it occurs as the root symbol on the left of a rule, otherwise $f$ is called constructor
- values are terms made up from constructors and variables

## Definition

we define the runtime complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dh}_{\mathcal{R}}(t) = \max\{n \mid \exists u\ t \to_{\mathcal{R}}^{n} u\}$$

$$\mathrm{rc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n \text{ and } t \text{ is basic}\}$$

a term $f(t_1, \ldots, t_n)$ is basic if $f$ is defined and all $t_i$ are values

## Example (continued)

we obtain $\mathrm{rc}_{\mathcal{R}} \in \mathrm{O}(n)$

# Historic Detour: Derivational Complexity Analysis

 says

thank you very much for the explanation, but is runtime complexity a natural notion for rewriting?

# Historic Detour: Derivational Complexity Analysis

 says

thank you very much for the explanation, but is runtime complexity a natural notion for rewriting?

## Example

consider the following set of rules

$$\mathsf{d}(\mathsf{g}(0,0),y) \to \mathsf{e}(0) \qquad \mathsf{h}(\mathsf{e}(x),y) \to \mathsf{h}(\mathsf{d}(x,y),\mathsf{s}(y))$$

$$\mathsf{d}(\mathsf{g}(x,y),z) \to \mathsf{g}(\mathsf{e}(x),\mathsf{d}(y,z)) \qquad \mathsf{g}(\mathsf{e}(x),\mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x,y))$$

$$\mathsf{d}(\mathsf{g}(\mathsf{g}(x,y),0),\mathsf{s}(z)) \to \mathsf{g}(\mathsf{d}(\mathsf{g}(x,y),\mathsf{s}(z)),\mathsf{d}(\mathsf{g}(x,y),z))$$

# Historic Detour: Derivational Complexity Analysis

 says

thank you very much for the explanation, but is runtime complexity a natural notion for rewriting?

## Example

consider the following set of rules

$$\mathsf{d}(\mathsf{g}(0,0),y) \to \mathsf{e}(0) \qquad \mathsf{h}(\mathsf{e}(x),y) \to \mathsf{h}(\mathsf{d}(x,y),\mathsf{s}(y))$$

$$\mathsf{d}(\mathsf{g}(x,y),z) \to \mathsf{g}(\mathsf{e}(x),\mathsf{d}(y,z)) \qquad \mathsf{g}(\mathsf{e}(x),\mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x,y))$$

$$\mathsf{d}(\mathsf{g}(\mathsf{g}(x,y),0),\mathsf{s}(z)) \to \mathsf{g}(\mathsf{d}(\mathsf{g}(x,y),\mathsf{s}(z)),\mathsf{d}(\mathsf{g}(x,y),z))$$

 says

ahem, actually no

## Definitions

- we define the derivational complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n\}$$

## Definitions

- we define the derivational complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n\}$$

- method X induces derivational complexity from class C if

$$\text{“}\mathcal{R} \text{ terminating by X”} \quad \text{implies} \quad \mathrm{dc}_{\mathcal{R}} \in \mathsf{C}$$

## Definitions

- we define the derivational complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n\}$$

- method X induces derivational complexity from class C if

$$\text{“}\mathcal{R} \text{ terminating by X”} \quad \text{implies} \quad \mathrm{dc}_{\mathcal{R}} \in \mathsf{C}$$

## Theorem

*the multiset path order induces primitive recursive derivational complexity and primitive recursive runtime complexity*

## Definitions

- we define the derivational complexity wrt. a terminating $\mathcal{R}$:

$$\mathrm{dc}_{\mathcal{R}}(n) = \max\{\mathrm{dh}_{\mathcal{R}}(t) \mid \text{size of } t \leqslant n\}$$

- method X induces derivational complexity from class C if

$$\text{“}\mathcal{R} \text{ terminating by X”} \quad \text{implies} \quad \mathrm{dc}_{\mathcal{R}} \in \mathrm{C}$$

## Theorem

*the multiset path order induces primitive recu*
*complexity and primitive recursive runtime co*


D. Hofbauer and C. Lautemann.
Termination proofs and the length of derivations.
In *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 167–177, 1989.

D. Hofbauer.
Termination proofs by multiset path orderings imply primitive recursive derivation lengths.
*TCS*, 105:129–140, 1992.

# The Multiset Path Order

let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

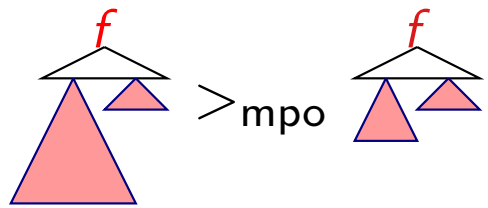
$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

# The Multiset Path Order

let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

2. $t = f(t_1, \ldots, t_n)$ and $\{\!\{s_1, \ldots, s_n\}\!\} >^{\mathsf{mul}}_{\mathsf{mpo}} \{\!\{t_1, \ldots, t_n\}\!\}$

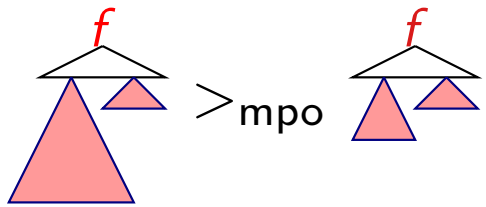


# The Multiset Path Order

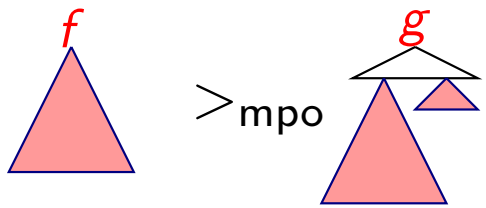
let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

2. $t = f(t_1, \ldots, t_n)$ and $\{\!\{s_1, \ldots, s_n\}\!\} >^{\mathsf{mul}}_{\mathsf{mpo}} \{\!\{t_1, \ldots, t_n\}\!\}$




3. $t = g(t_1, \ldots, t_m)$ with $f > g$ and $\forall i\ s >_{\mathsf{mpo}} t_i$

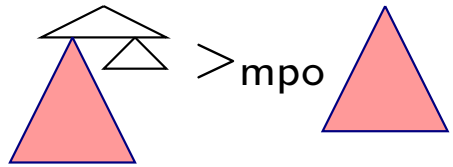


# The Multiset Path Order

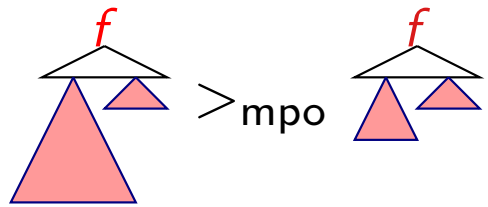
let $>$ denote a precedence; $>$ induces order $>_{\mathsf{mpo}}$:

$s = f(s_1, \ldots, s_n) >_{\mathsf{mpo}} t$ if either

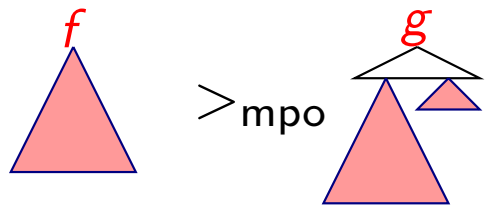
1. $\exists i\ s_i \geqslant_{\mathsf{mpo}} t$




2. $t = f(t_1, \ldots, t_n)$ and
   $\{\!\{s_1, \ldots, s_n\}\!\} >^{\mathsf{mul}}_{\mathsf{mpo}} \{\!\{t_1, \ldots, t_n\}\!\}$




3. $t = g(t_1, \ldots, t_m)$ with $f > g$ and
   $\forall i\ s >_{\mathsf{mpo}} t_i$



## Theorem

- *polynomial interpretations induce double-exponential derivational complexity*
- *lexicographic path orders induce multiple recursive functions*
- *DP framework in conjunction with the subterm criterion induces multiple recursive functions*
- *simplification orders induce functions elementary in* $\mathsf{H}_\Lambda$*, where* $\Lambda \ggg \epsilon_0$

## Theorem

- *polynomial interpretations induce double-exponential derivational complexity*
- *lexicographic path orders induce multiple recursive functions*
- *DP framework in conjunction with the subterm criterion induces multiple recursive functions*
- *simplification orders induce functions elementary in* $\mathsf{H}_\Lambda$*, where* $\Lambda \ggg \epsilon_0$
- *...*

## Theorem

- *polynomial interpretations induce double-exponential derivational complexity*
- *lexicographic path orders induce multiple recursive functions*
- *DP framework in conjunction with the subterm criterion induces multiple recursive functions*
- *simplification orders induce functions elementary in* $\mathsf{H}_\Lambda$*, where* $\Lambda \ggg \epsilon_0$
- *...*


I. Lepper.
Simply terminating rewrite systems with long derivations.
*Arch. Math. Logic*, 43:1–18, 2004.

A. Weiermann.
Complexity bounds for some finite forms of Kruskal's theorem.
*JSC*, 18(5):463–488, November 1994.

## Example (revisited)

$$\mathsf{d}(\mathsf{g}(0,0),y) \to \mathsf{e}(0) \qquad \mathsf{h}(\mathsf{e}(x),y) \to \mathsf{h}(\mathsf{d}(x,y),\mathsf{s}(y))$$

$$\mathsf{d}(\mathsf{g}(x,y),z) \to \mathsf{g}(\mathsf{e}(x),\mathsf{d}(y,z)) \qquad \mathsf{g}(\mathsf{e}(x),\mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x,y))$$

$$\mathsf{d}(\mathsf{g}(\mathsf{g}(x,y),0),\mathsf{s}(z)) \to \mathsf{g}(\mathsf{d}(\mathsf{g}(x,y),\mathsf{s}(z)),\mathsf{d}(\mathsf{g}(x,y),z))$$

## Example (revisited)

$$\mathsf{d}(\mathsf{g}(0,0),y) \to \mathsf{e}(0) \qquad \mathsf{h}(\mathsf{e}(x),y) \to \mathsf{h}(\mathsf{d}(x,y),\mathsf{s}(y))$$

$$\mathsf{d}(\mathsf{g}(x,y),z) \to \mathsf{g}(\mathsf{e}(x),\mathsf{d}(y,z)) \qquad \mathsf{g}(\mathsf{e}(x),\mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x,y))$$

$$\mathsf{d}(\mathsf{g}(\mathsf{g}(x,y),0),\mathsf{s}(z)) \to \mathsf{g}(\mathsf{d}(\mathsf{g}(x,y),\mathsf{s}(z)),\mathsf{d}(\mathsf{g}(x,y),z))$$

## The Hydra Battle by Kirby and Paris

1. the beast is a finite tree, each leaf corresponds to a head; Hercules chops off heads of the Hydra, but the Hydra regrows:
   - if the cut head has a pre-predecessor, then the remaining subtree issued from this node is multiplied by the stage of the game.
   - otherwise the Hydra ignores the loss.
2. Hercules wins, when the beast is reduced to the empty tree.

## Example (revisited)

$$\mathsf{d}(\mathsf{g}(0,0),y) \to \mathsf{e}(0) \qquad \mathsf{h}(\mathsf{e}(x),y) \to \mathsf{h}(\mathsf{d}(x,y),\mathsf{s}(y))$$

$$\mathsf{d}(\mathsf{g}(x,y),z) \to \mathsf{g}(\mathsf{e}(x),\mathsf{d}(y,z)) \qquad \mathsf{g}(\mathsf{e}(x),\mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x,y))$$

$$\mathsf{d}(\mathsf{g}(\mathsf{g}(x,y),0),\mathsf{s}(z)) \to \mathsf{g}(\mathsf{d}(\mathsf{g}(x,y),\mathsf{s}(z)),\mathsf{d}(\mathsf{g}(x,y),z))$$

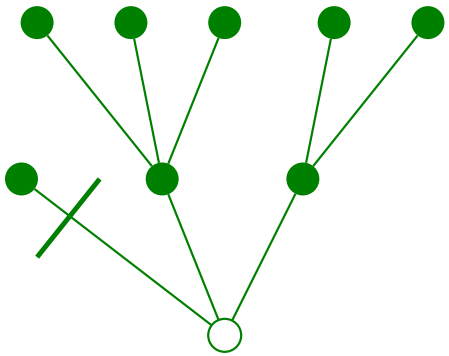
## The Hydra Battle by Kirby and Paris

1. the beast is a finite tree, each leaf corresponds to a head; Hercules chops off heads of the Hydra, but the Hydra regrows:
   - if the cut head has a pre-predecessor, then the remaining subtree issued from this node is multiplied by the stage of the game.
   - otherwise the Hydra ignores the loss.
2. Hercules wins, when the beast is reduced to the empty tree.

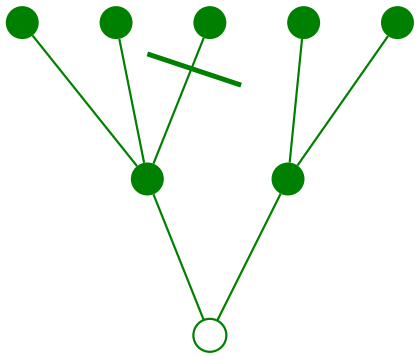
## Theorem

*termination of the battle is independent: PA $\nvdash$ the battle terminates*

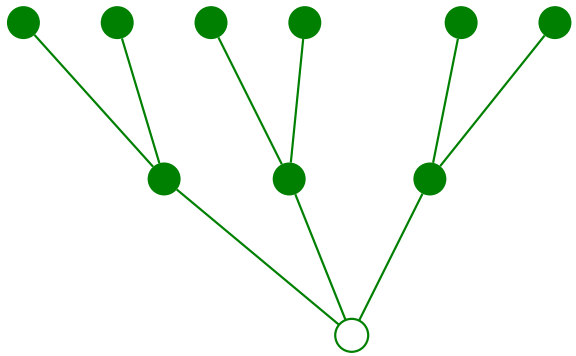
## Example



$(H_1, 1)$

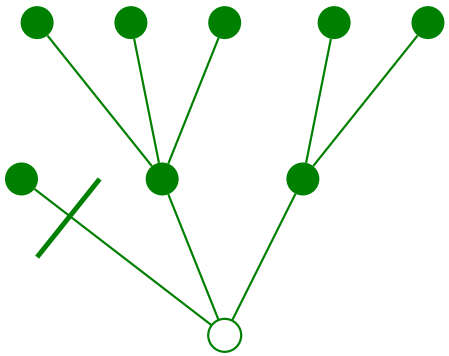


$(H_2, 2)$

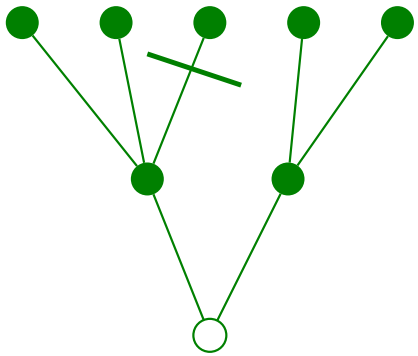


$(H_3, 3)$

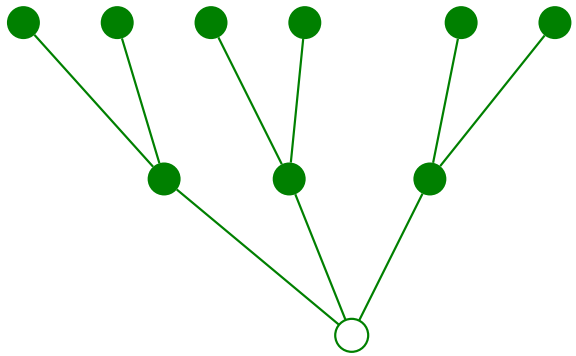
## Example



$(\omega^3 \oplus \omega^2 \oplus 1, 1)$

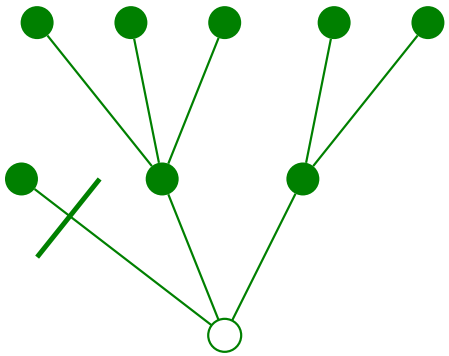


$(\omega^3 \oplus \omega^2, 2)$



$(\omega^2 \cdot 3, 3)$

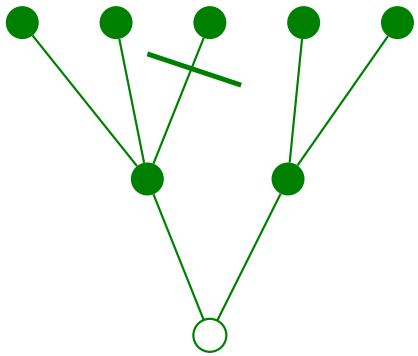
## Example



$(\omega^3 \oplus \omega^2 \oplus 1, 1)$

$\omega^3 + \omega^2 + 1$

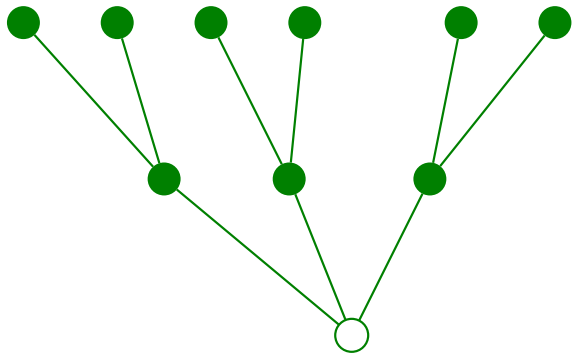
$>$



$(\omega^3 \oplus \omega^2, 2)$

$\omega^3 + \omega^2$

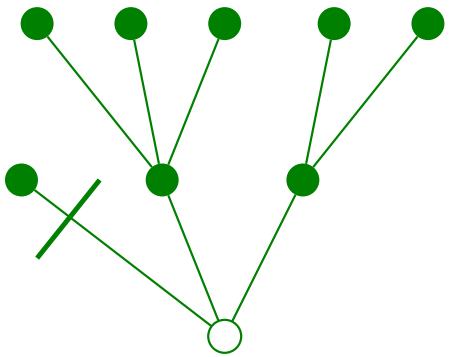
$>$



$(\omega^2 \cdot 3, 3)$

$\omega^2 \cdot 3$

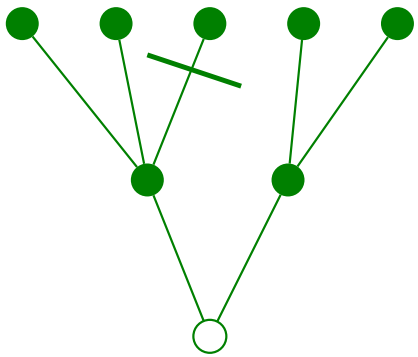
## Example



$(\omega^3 \oplus \omega^2 \oplus 1, 1)$

$\mathsf{H}_{\omega^3+\omega^2+1}$

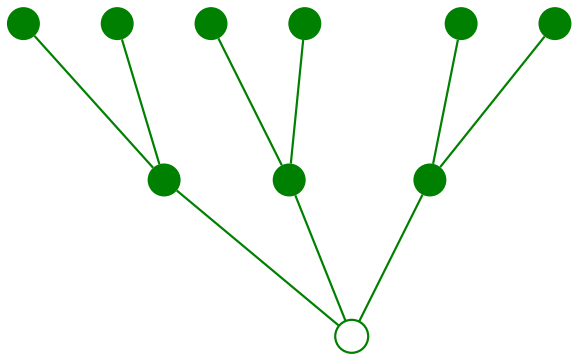
$\succ_{ed}$



$(\omega^3 \oplus \omega^2, 2)$

$\mathsf{H}_{\omega^3+\omega^2}$

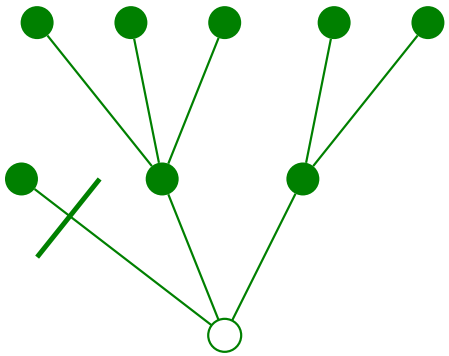
$\succ_{ed}$



$(\omega^2 \cdot 3, 3)$

$\mathsf{H}_{\omega^2 \cdot 3}$

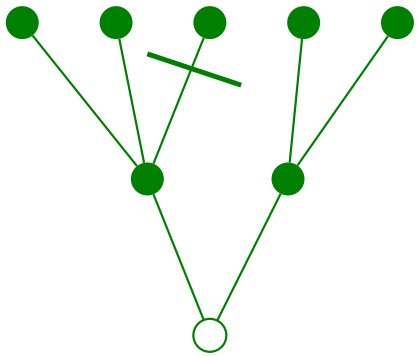
## Example



$(\omega^3 \oplus \omega^2 \oplus 1, 1)$

$\mathsf{H}_{\omega^3+\omega^2+1}$

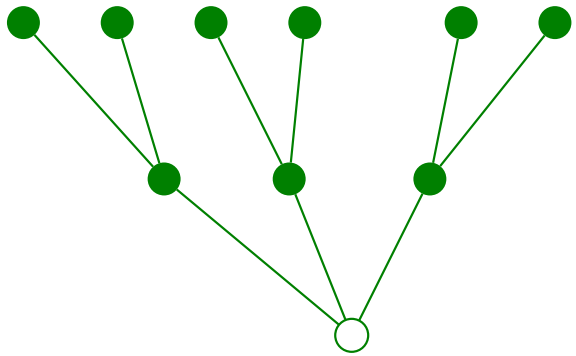
$\succ_{ed}$



$(\omega^3 \oplus \omega^2, 2)$

$\mathsf{H}_{\omega^3+\omega^2}$

$\succ_{ed}$



$(\omega^2 \cdot 3, 3)$

$\mathsf{H}_{\omega^2 \cdot 3}$

## Remark

- based on the game, Dershowitz and Jouannaud designed the TRS as a challenging termination problem
- however, the implementation is buggy and has later been rectified by Dershowitz

## Example

$\mathsf{h}(\mathsf{e}(x), y) \to \mathsf{h}(\mathsf{d}(x, y), \mathsf{s}(y))$ $\qquad (\alpha, n) \Longrightarrow (\alpha_n, n+1)$

$\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), 0) \to \mathsf{e}(y)$ $\qquad$ strategy of the game

$\mathsf{d}(\mathsf{g}(0, x), y) \to \mathsf{e}(x)$

$\mathsf{d}(\mathsf{g}(x, y), z) \to \mathsf{g}(\mathsf{d}(x, z), \mathsf{e}(y))$

$\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), \mathsf{s}(z)) \to \mathsf{g}(\mathsf{e}(x), \mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), z))$

$\mathsf{g}(\mathsf{e}(x), \mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x, y))$ $\qquad$ auxiliary rule

## Remark

- based on the game, Dershowitz and Jouannaud designed the TRS as a challenging termination problem
- however, the implementation is buggy and has later been rectified by Dershowitz

## Example

$\mathsf{h}(\mathsf{e}(x), y) \to \mathsf{h}(\mathsf{d}(x, y), \mathsf{s}(y))$

$\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), 0) \to \mathsf{e}(y)$

$\mathsf{d}(\mathsf{g}(0, x), y) \to \mathsf{e}(x)$

$\mathsf{d}(\mathsf{g}(x, y), z) \to \mathsf{g}(\mathsf{d}(x, z), \mathsf{e}(y))$

$\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), \mathsf{s}(z)) \to \mathsf{g}(\mathsf{e}(x), \mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), z))$

$\mathsf{g}(\mathsf{e}(x), \mathsf{e}(y)) \to \mathsf{e}(\mathsf{g}(x, y))$

$(\alpha, n) \Longrightarrow (\alpha_n, n + 1)$

strategy of the game

## Remark

- based on the game, Dershowitz and Jouannaud designed the TRS as a challenging termination problem
- however, the implementation is buggy and has later been rectified by Dershowitz

## RTALooP # 23

*Must any termination ordering used for proving termination of the Battle of Hydra and Hercules-system have the Howard[-Bachmann] ordinal as its order type?*[a]

[a] `http://www.win.tue.nl/rtaloop/`.

## RTALooP # 23

*Must any termination ordering used for proving termination of the Battle of Hydra and Hercules-system have the Howard[-Bachmann] ordinal as its order type?*[a]

[a] `http://www.win.tue.nl/rtaloop/`.

## Cichon's Conjecture

The derivational complexity induced by any termination order of order type $\alpha$ is bounded by the slow-growing hierarchy indexed by $\alpha$

## RTALooP # 23

*Must any termination ordering used for proving termination of the Battle of Hydra and Hercules-system have the Howard[-Bachmann] ordinal as its order type?*[a]

[a] `http://www.win.tue.nl/rtaloop/`.

## Cichon's Conjecture

The derivational complexity induced by any termination order of order type $\alpha$ is bounded by the slow-growing hierarchy indexed by $\alpha$

## Answer

No.

## RTALooP # 23

*Must any termination ordering used for proving termination of the Battle of Hydra and Hercules-system have the Howard[-Bachmann] ordinal as its order type?*[a]

[a] `http://www.win.tue.nl/rtaloop/`.

## Cichon's Conjecture

The derivational complexity induced by any termination order of order type $\alpha$ is bounded by the slow-growing hierarchy indexed by $\alpha$

## Answer

No.

GM.
The Hydra battle and Cichon's principle.
*AAECC*, 20(2):133–158, 2009.

says

aha, coming back to runtime complexity, I found the following . . .

Tyrolean Complexity Tool - Web Interface - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Tyrolean Complexity Tool

colo6-c703.uibk.ac.at/tct/tct-trs/#output

CBR Home | TcT Home | Download | Experiments | TCT Web

**Input** *(in xml or trs format)*

select example ... | or upload file | Browse... | No file selected.

```
1 (VAR x xs acc)
2
3 (RULES
4   rev(xs) -> revacc(xs,nil)
5   revacc(nil,acc) -> acc
6   revacc(cons(x,xs),acc) -> revacc(xs,cons(x,acc))
7 )
```

**Category**

Complexity Measure: (•) Runtime Complexity ( ) Derivational Complexity

Rewriting Strategy: (•) Full Rewriting ( ) Innermost Rewriting

**Search Strategy**

(•) Automatic ( ) Certify ( ) Customised by user

check with timeout of 30 seconds.

**Output**

```
WORST_CASE(?,O(n^1))
*** 1 Success [(?,O(n^1))]  ***
    Considered Problem:
      Strict DP Rules:
```

 says

indeed, TCT can fully automatically handle the TRS, for example using a method called matrix interpretations

 says

indeed, TCT can fully automatically handle the TRS, for example using a method called matrix interpretations

## Definition

a matrix interpretation of dimension $n$ is a pair $\mathcal{M} = (\mathbb{N}^n, >)$ such that

1. $\forall\ f \in \mathcal{F}$:

$$f_{\mathcal{M}}(\vec{v}_1, \ldots, \vec{v}_k) = M_1\vec{v}_1 + \ldots + M_k\vec{v}_k + \vec{f}$$

2. the $M_i$ are square matrices such that $(M_i)_{1,1} \geqslant 1$
3. the order $>$ is defined as:

$$(x_1, x_2, \ldots, x_n)^\top > (y_1, y_2, \ldots, y_n)^\top$$

if $x_1 > y_1$ and $\forall\ i \geqslant 2$: $x_i \geqslant y_i$

## Example

consider the TRS $(x \circ y) \circ z \to x \circ (y \circ z)$ together with the matrix interpretation $\mathcal{M}$

$$\circ_{\mathcal{M}}(\vec{x}, \vec{y}) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \vec{y} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

## Example

consider the TRS $(x \circ y) \circ z \to x \circ (y \circ z)$ together with the matrix interpretation $\mathcal{M}$

$$\circ_{\mathcal{M}}(\vec{x}, \vec{y}) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \vec{y} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

we obtain $\forall$ assignments to the variables:

$$[(x \circ y) \circ z)] > [(x \circ (y \circ z))]$$

we say $\mathcal{M}$ is compatible with the TRS

## Example

consider the TRS $(x \circ y) \circ z \to x \circ (y \circ z)$ together with the matrix interpretation $\mathcal{M}$

$$\circ_{\mathcal{M}}(\vec{x}, \vec{y}) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \vec{y} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

we obtain $\forall$ assignments to the variables:

$$[(x \circ y) \circ z)] > [(x \circ (y \circ z))]$$

we say $\mathcal{M}$ is compatible with the TRS

## Observation

$$t = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} t_4 \to_{\mathcal{R}} \cdots$$

implies

$$[t] > [t_1] > [t_2] > [t_3] > [t_4] > \cdots$$

for compatible matrix interpretations $\mathcal{M}$

## Definition

let $\mathcal{M}$ denote a matrix interpretation and let

$$M = \max\{M_i \mid M_i \text{ a square matrix used in } f_{\mathcal{M}}\}$$

then $M$ is called maximums matrix of $\mathcal{M}$

## Definition

let $\mathcal{M}$ denote a matrix interpretation and let

$$M = \max\{M_i \mid M_i \text{ a square matrix used in } f_{\mathcal{M}}\}$$

then $M$ is called maximums matrix of $\mathcal{M}$

## Theorem

*let $\mathcal{M}$ be a matrix interpretation of dimension $d$ compatible with a TRS $\mathcal{R}$ and let $M$ be the maximum matrix of $\mathcal{M}$, if the spectral radius $\rho(M) \leqslant 1$, then $\mathrm{rc}_{\mathcal{R}} \in \mathrm{O}(n^d)$*

## Proof.

by linear algebra, for example Jordan Normal Form Theorem ■

## Definition

let $\mathcal{M}$ denote a matrix interpretation and let

$$M = \max\{M_i \mid M_i \text{ a square matrix used in } f_{\mathcal{M}}\}$$

then $M$ is called maximums matrix of $\mathcal{M}$

## Theorem

*let $\mathcal{M}$ be a matrix interpretation of dimension $d$ compatible with a TRS $\mathcal{R}$ and let $M$ be the maximum matrix of $\mathcal{M}$, if the spectral radius $\rho(M) \leqslant 1$, then $\mathrm{rc}_{\mathcal{R}} \in \mathrm{O}(n^d)$*

## Proof.

by linear algebra, for example Jordan Normal Form Theorem ■

A. Middeldorp, GM, F. Neurauter, J. Waldmann, and H. Zankl.
Joint spectral radius theory for automated complexity analysis of rewrite systems.
In *Proc. 4th CAI*, volume 6742 of *LNCS*, pages 1–20, 2011.

## Definition

let $\mathcal{M}$ denote a matrix interpretation and let

$$M = \max\{M_i \mid M_i \text{ a square matrix used in } f_{\mathcal{M}}\}$$

then $M$ is called maximums matrix of $\mathcal{M}$

## Theorem

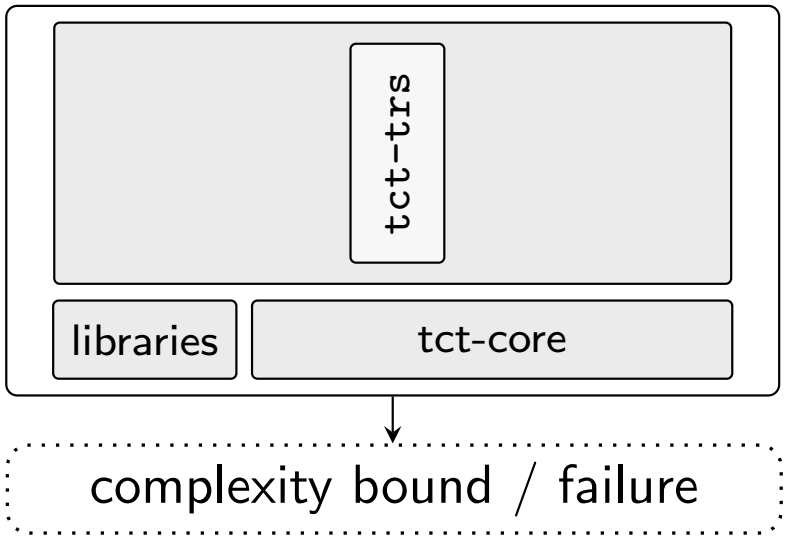
*let $\mathcal{M}$ be a matrix interpretation of dim... a TRS $\mathcal{R}$ and let $M$ be the maximum matrix ... s $\rho(M) \leqslant 1$, then* $\mathrm{rc}_{\mathcal{R}} \in \mathrm{O}(n^d)$

## Proof.

by linear algebra, for example Jordan Normal Form Theorem ■

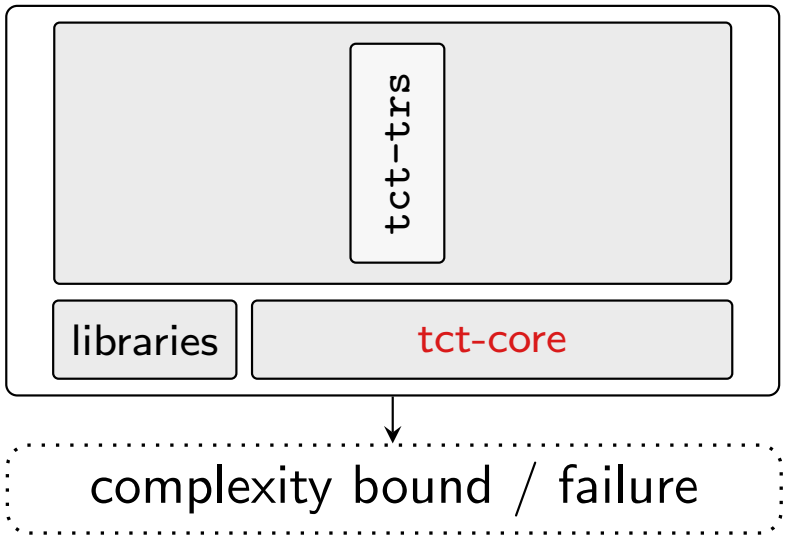
A. Middeldorp, GM, F. Neurauter, J. Waldmann, and H. Zankl.
Joint spectral radius theory for automated complexity analysis of rewrite systems.
In *Proc. 4th CAI*, volume 6742 of *LNCS*, pages 1–20, 2011.

# Tyrolean Complexity Tool (`tct-trs`)



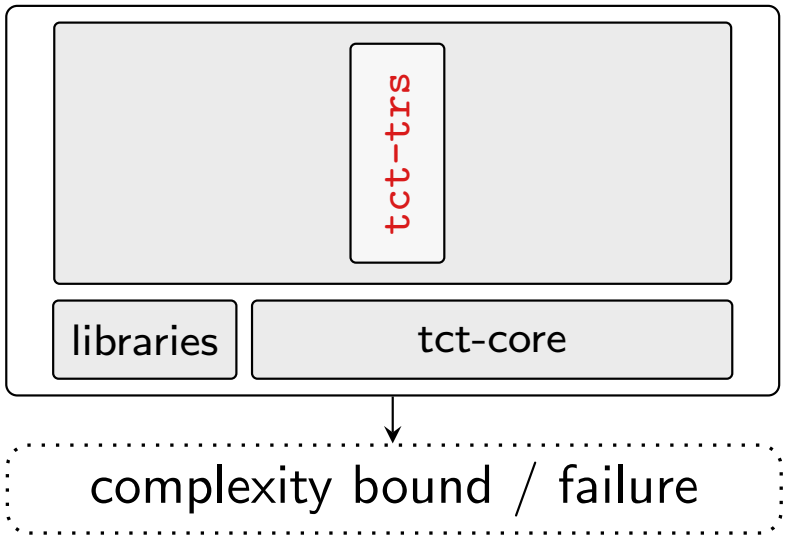

- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

# Tyrolean Complexity Tool (`tct-trs`)



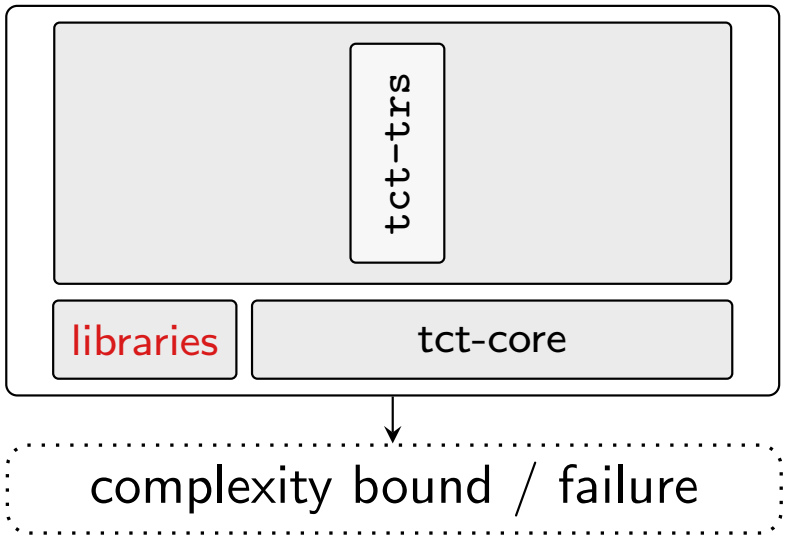

- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

# Tyrolean Complexity Tool (`tct-trs`)



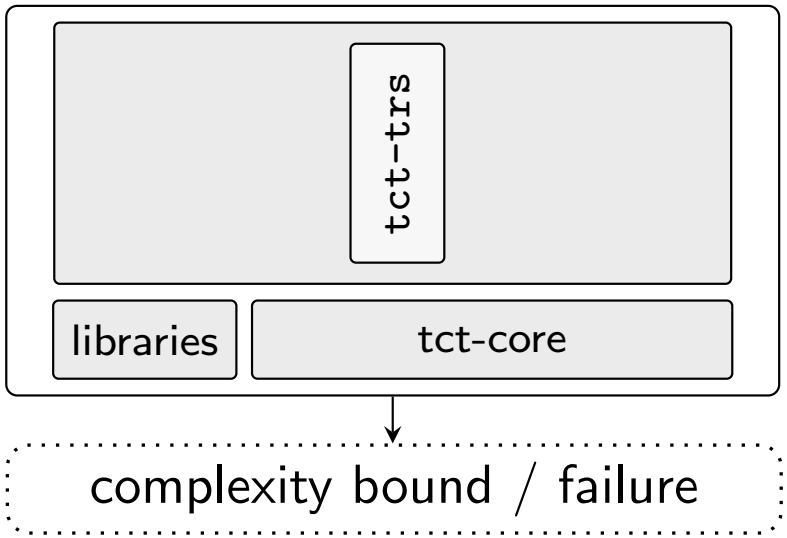

- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

# Tyrolean Complexity Tool (`tct-trs`)




- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

# Tyrolean Complexity Tool (tct-trs)




- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in t… …es
- m… …
- p…

M. Avanzini, C. Sternagel, and R. Thiemann.
Certification of complexity proofs using CeTA
In *Proc. 26th RTA*, volume 36 of *LIPIcs*, pag…

M. Avanzini and GM.
A combination framework for complexity.
*IC*, 248:22–55, 2016.

M. Avanzini, GM, and M. Schaper.
Tct: Tyrolean complexity tool.
In *Proc. 22nd TACAS*, volume 9636 of *LNCS*, pages 407–423, 2016.


```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil})$
$\mathsf{rev}'(\mathsf{nil}, acc) \to acc$
$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$


```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

$\mathrm{rev}(xs) \to \mathrm{rev}'(xs, \mathrm{nil})$
$\mathrm{rev}'(\mathrm{nil}, acc) \to acc$
$\mathrm{rev}'(x :: xs, acc) \to \mathrm{rev}'(xs, x :: acc)$


$\mathrm{rev}(xs) \to \mathrm{rev}'(xs, \mathrm{nil})$
$\mathrm{rev}'(\mathrm{nil}, acc) \to acc$
$\mathrm{rev}'(x :: xs, acc) \to \mathrm{rev}'(xs, x :: acc)$

$O(n)$


```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil})$
$\mathsf{rev}'(\mathsf{nil}, acc) \to acc$
$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$


$\mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil})$
$\mathsf{rev}'(\mathsf{nil}, acc) \to acc$
$\mathsf{rev}'(x :: xs, acc) \to \mathsf{rev}'(xs, x :: acc)$

$O(n)$

## Definition

- a transformation is complexity reflecting, if upper bound is reflected
- complexity preserving, if lower bound is preserved

# Example Run of tct-hoca

- textbook example `rev`

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

# Example Run of tct-hoca

- textbook example `rev`

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

- defunctionalised applicative rewrite system

$$
\begin{aligned}
\mathsf{main}(x_0) &\to \mathsf{m}_1(x_0)\ @\ \mathsf{f} & \mathsf{r}(x_0)\ @\ x_1 &\to x_0\ @\ \mathsf{r}_1\ @\ [\,]\ @\ x_1\\
\mathsf{m}_1(x_0)\ @\ x_1 &\to \mathsf{m}_2(x_0)\ @\ \mathsf{r}(x_1) & \mathsf{r}_1\ @\ x_0 &\to \mathsf{r}_2(x_0)\\
\mathsf{m}_2(x_0)\ @\ x_1 &\to x_1\ @\ x_0 & \mathsf{r}_2(x_0)\ @\ x_1 &\to x_1 :: x_0\\
\mathsf{f}\ @\ x_0 &\to \mathsf{f}_1\ @\ x_0 & \mathsf{f}_3(x_0, x_1)\ @\ x_2 &\to \mathsf{f}_4(x_2, x_0, x_1)\\
\mathsf{f}_1\ @\ x_1 &\to \mathsf{f}_2(x_1) & \mathsf{f}_4([\,], x_0, x_1) &\to x_1\\
\mathsf{f}_2(x_1)\ @\ x_2 &\to \mathsf{f}_3(x_1, x_2) & \mathsf{f}_4(x_0 :: x_1, x_2, x_3) &\to \mathsf{f}\ @\ x_1\ @\ (x_2\ @\ x_3\ @\ x_0)\ @\ x_2
\end{aligned}
$$

# Example Run of tct-hoca

- textbook example `rev`

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

- defunctionalised applicative rewrite system

$$\begin{aligned}
\mathsf{main}(x_0) &\to \mathsf{m}_1(x_0)\ @\ \mathsf{f} & \mathsf{r}(x_0)\ @\ x_1 &\to x_0\ @\ \mathsf{r}_1\ @\ [\,]\ @\ x_1\\
\mathsf{m}_1(x_0)\ @\ x_1 &\to \mathsf{m}_2(x_0)\ @\ \mathsf{r}(x_1) & \mathsf{r}_1\ @\ x_0 &\to \mathsf{r}_2(x_0)\\
\mathsf{m}_2(x_0)\ @\ x_1 &\to x_1\ @\ x_0 & \mathsf{r}_2(x_0)\ @\ x_1 &\to x_1 :: x_0\\
\mathsf{f}\ @\ x_0 &\to \mathsf{f}_1\ @\ x_0 & \mathsf{f}_3(x_0, x_1)\ @\ x_2 &\to \mathsf{f}_4(x_2, x_0, x_1)\\
\mathsf{f}_1\ @\ x_1 &\to \mathsf{f}_2(x_1) & \mathsf{f}_4([\,], x_0, x_1) &\to x_1\\
\mathsf{f}_2(x_1)\ @\ x_2 &\to \mathsf{f}_3(x_1, x_2) & \mathsf{f}_4(x_0 :: x_1, x_2, x_3) &\to \mathsf{f}\ @\ x_1\ @\ (x_2\ @\ x_3\ @\ x_0)\ @\ x_2
\end{aligned}$$

- Simplified first-order term rewrite system

$$\mathsf{main}(x_0) \to \mathsf{f}(\mathsf{nil}, x_0) \qquad \mathsf{f}(x_0, \mathsf{nil}) \to x_0 \qquad \mathsf{f}(x_0, x_1 :: x_2) \to \mathsf{f}(x_1 :: x_0, x_2)$$

# Example Run of tct-hoca

- textbook example `rev`

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

- defunctionalised applicative rewrite system

$$\begin{aligned}
\mathsf{main}(x_0) &\to \mathsf{m}_1(x_0) \mathbin{@} \mathsf{f} & \mathsf{r}(x_0) \mathbin{@} x_1 &\to x_0 \mathbin{@} \mathsf{r}_1 \mathbin{@} [\,] \mathbin{@} x_1\\
\mathsf{m}_1(x_0) \mathbin{@} x_1 &\to \mathsf{m}_2(x_0) \mathbin{@} \mathsf{r}(x_1) & \mathsf{r}_1 \mathbin{@} x_0 &\to \mathsf{r}_2(x_0)\\
\mathsf{m}_2(x_0) \mathbin{@} x_1 &\to x_1 \mathbin{@} x_0 & \mathsf{r}_2(x_0) \mathbin{@} x_1 &\to x_1 :: x_0\\
\mathsf{f} \mathbin{@} x_0 &\to \mathsf{f}_1 \mathbin{@} x_0 & \mathsf{f}_3(x_0, x_1) \mathbin{@} x_2 &\to \mathsf{f}_4(x_2, x_0, x_1)\\
\mathsf{f}_1 \mathbin{@} x_1 &\to \mathsf{f}_2(x_1) & \mathsf{f}_4([\,], x_0, x_1) &\to x_1\\
\mathsf{f}_2(x_1) \mathbin{@} x_2 &\to \mathsf{f}_3(x_1, x_2) & \mathsf{f}_4(x_0 :: x_1, x_2, x_3) &\to \mathsf{f} \mathbin{@} x_1 \mathbin{@} (x_2 \mathbin{@} x_3 \mathbin{@} x_0) \mathbin{@} x_2
\end{aligned}$$

- Simplified first-order term rewrite system

$$\mathsf{main}(x_0) \to \mathsf{f}(\mathsf{nil}, x_0) \qquad \mathsf{f}(x_0, \mathsf{nil}) \to x_0 \qquad \mathsf{f}(x_0, x_1 :: x_2) \to \mathsf{f}(x_1 :: x_0, x_2)$$

says

the runtime complexity of `rev` is $O(n)$ . . . if TCT is sound

# Experimental Evidence

| | constant | linear | quadratic | polynomial | terminating |
|---|---|---|---|---|---|
| # systems | 2 | 14 | 18 | 20 | 25 |
| HoCA time | 4.56 | 4.56 | 4.56 | 4.56 | 6.48 |
| FOP time | 0.79 | 14.00 | 30.12 | 60.10 | 3.43 |

# Experimental Evidence

| | constant | linear | quadratic | polynomial | terminating |
|---|---|---|---|---|---|
| # systems | 2 | 14 | 18 | 20 | 25 |
| HoCA time | 4.56 | 4.56 | 4.56 | 4.56 | 6.48 |
| FOP time | 0.79 | 14.00 | 30.12 | 60.10 | 3.43 |

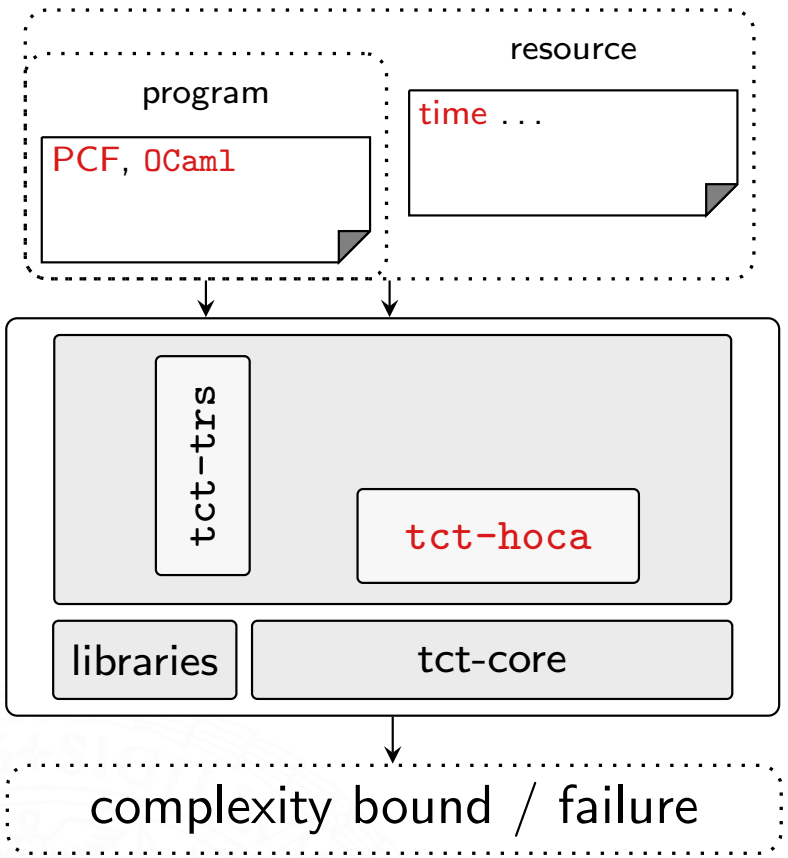
## Testbed

comprises 25 examples, for example including

1. `rev`, `foldl`, `map`, ...
2. merge-sort using a higher-order divide-and-conquer combinator
3. simple parsers relying on the monadic parser-combinator outlined in Okasaki's functional pearl

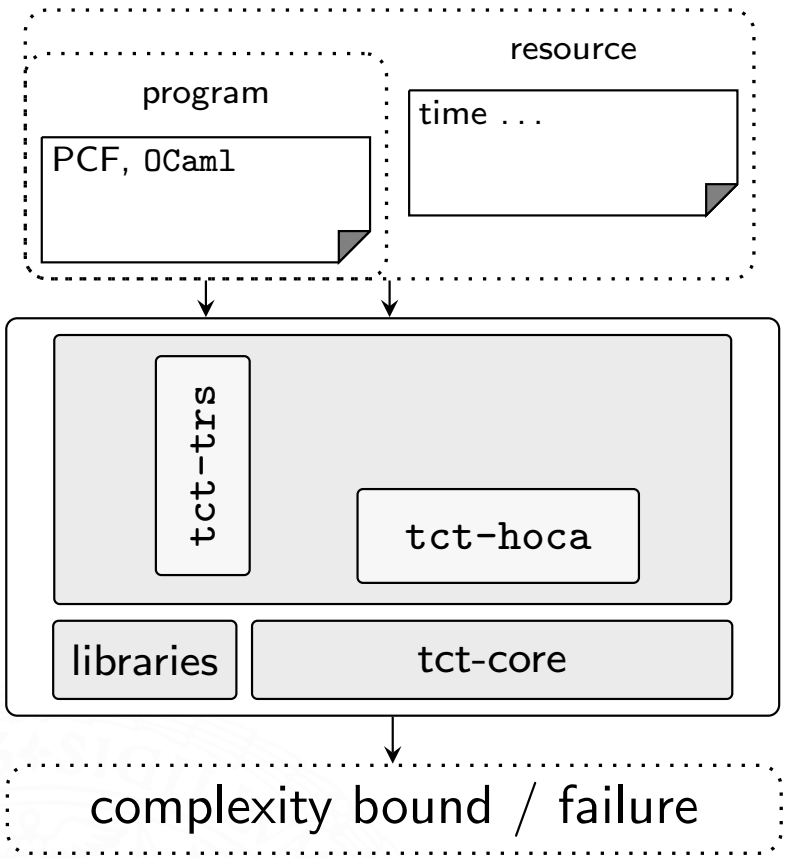
complexity is tested with TcT; termination with TTT2

# Tyrolean Complexity Tool (`tct-trs`, `tct-hoca`)




- fully automated resource analysis tool
- open source under BSD3
- implemented in Haskell
- competitive for higher-order functional programs
- employs SMT-solvers like `minismt` or Z3

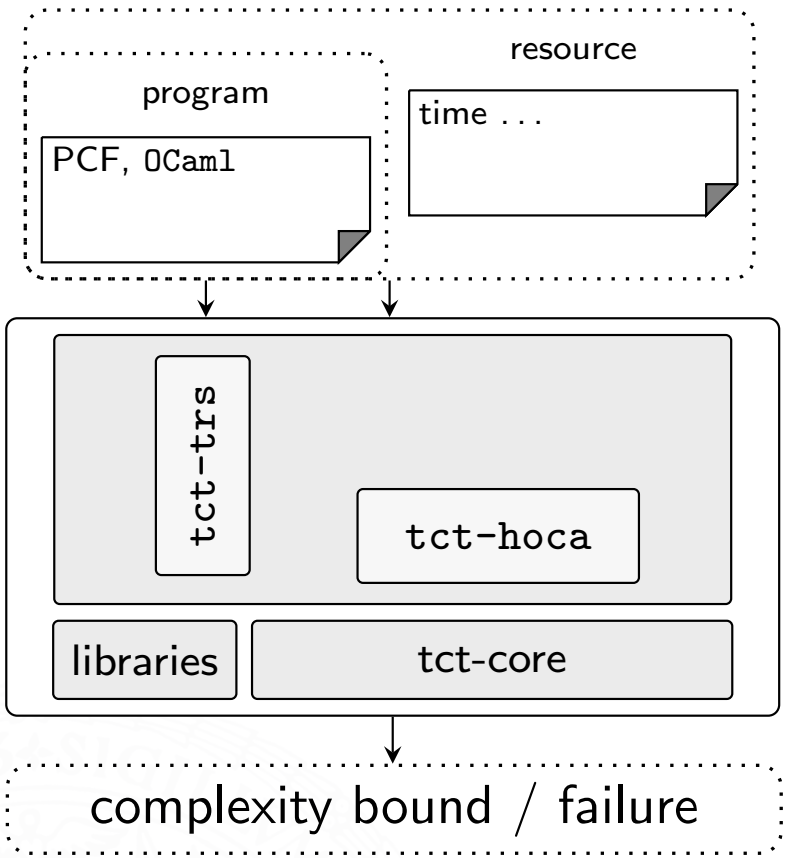
# Tyrolean Complexity Tool (`tct-trs`, `tct-hoca`)




- fully automated resource analysis tool
- open source under BSD3
- implemented in Haskell
- competitive for higher-order functional programs
- employs SMT-solvers like `minismt` or Z3

M. Avanzini, U. Dal Lago, and GM.
Analysing the complexity of functional programs: higher-order meets first-order.
In *Proc. 20th ICFP*, pages 152–164. ACM, 2015.

# Tyrolean Complexity Tool (tct-trs, tct-hoca)




- fully automated resource analysis tool
- open source under BSD3
- implemented in Haskell
- compet
  functio
- employ
  minisn

M. Avanzini, U. Dal Lago, and GM.
Analysing the complexity of functional programs: higher-order meets first-order.
In *Proc. 20th ICFP*, pages 152–164. ACM, 2015.

 says

this is all very exciting, but what about real programs?

 says

this is all very exciting, but what about real programs?

 says

well, if you insist . . .

 says

this is all very exciting, but what about real programs?

 says

well, if you insist . . .

## Example

```
public static void test(int n, int m){
  if (0 < n && n < m) {
    int j = n+1;
    while(j < n || j > n){
      if (j > m){
        j=0;
      } else {
        j=j+1;
    }}}}
```

## Remarks

- suppose $n < m$: while loop is executed as long as $j \neq n$ holds
- automation requires disjunctive bounds

## Remarks

- suppose $n < m$: while loop is executed as long as $j \neq n$ holds
- automation requires disjunctive bounds

## Definition

integer transition systems (aka transition systems) are sets of rules

- restricted to shallow terms consisting of function symbols and only variables as arguments
- constraints on variables

## Remarks

- suppose $n < m$: while loop is executed as long as $j \neq n$ holds
- automation requires disjunctive bounds

## Definition

integer transition systems (aka transition systems) are sets of rules

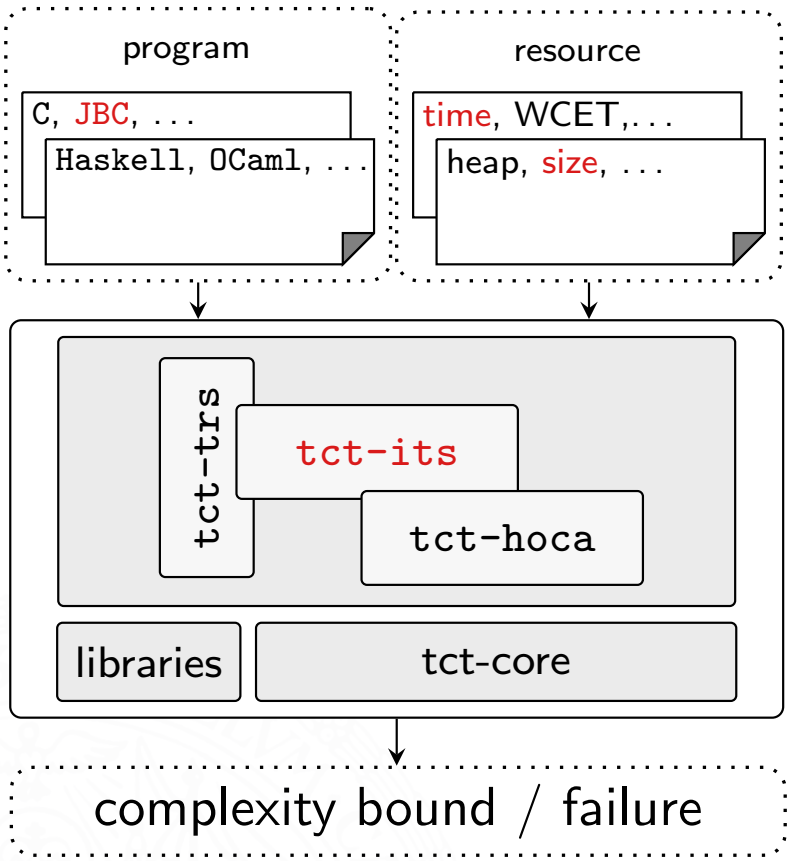
- restricted to shallow terms consisting of function symbols and only variables as arguments
- constraints on variables

## Example

```
start(m,n,j) → while(m,n,n+1) :|: m > n && n > 0
while(m,n,j) → while(m,n,0)   :|: n > 0 && j > n && j > m
while(m,n,j) → while(m,n,j+1) :|: n > 0 && j > n && j ≤ m
while(m,n,j) → while(m,n,j+1) :|: n > 0 && j < n && j ≤ m
```

# Uniform Resource Analysis by Rewriting

## TcT

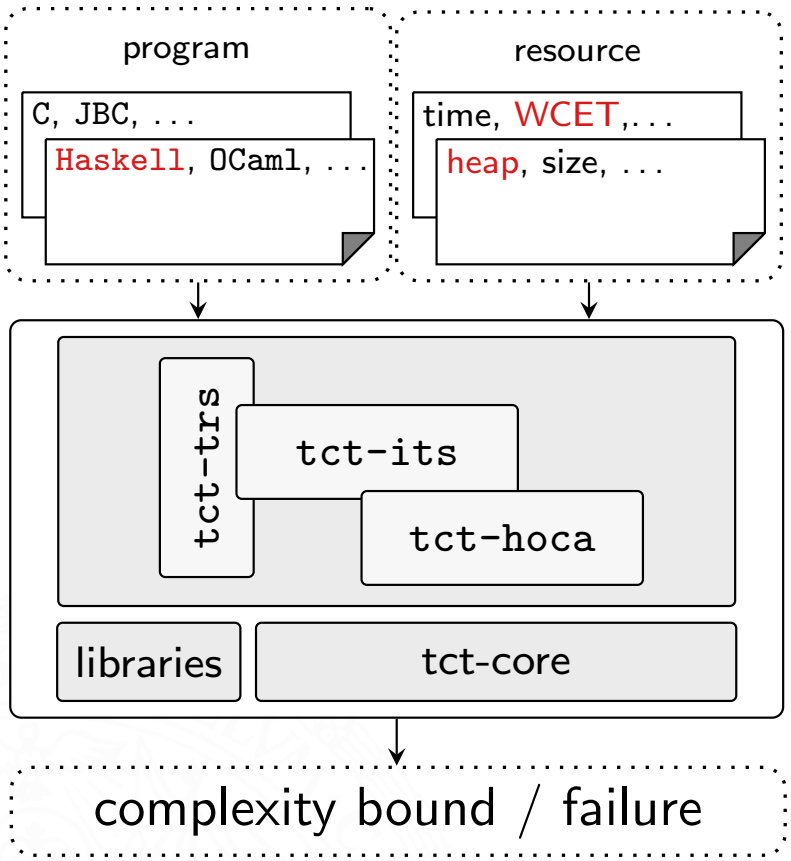



- fully automated resource analysis tool
- competitive for higher-order functional programs, occasionally competitive for object-oriented bytecode programs
- intertwined resource analysis framework

GM and M. Schaper.
From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation.
*IC*. To appear.

# Uniform Resource Analysis by Rewriting

## TcT

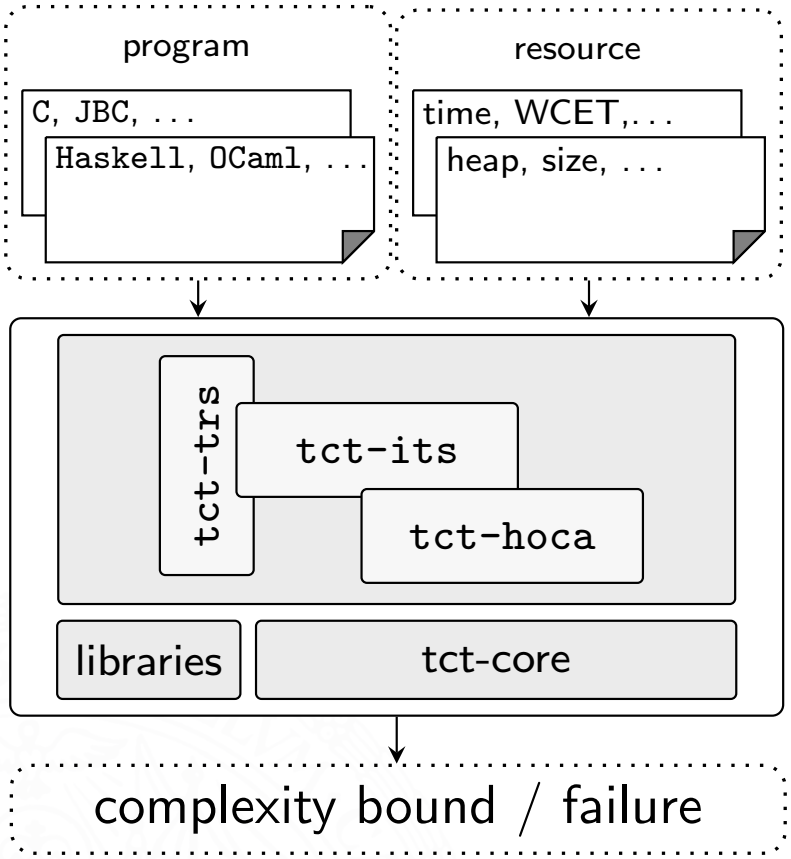



- fully automated resource analysis tool
- competitive for higher-order functional programs, occasionally competitive for object-oriented bytecode programs
- intertwined resource analysis framework

GM and M. Schaper.
From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation.
*IC*. To appear.

# Uniform Resource Analysis

## TcT, AProVE, CiaoPP, COSTA, SACO, ...

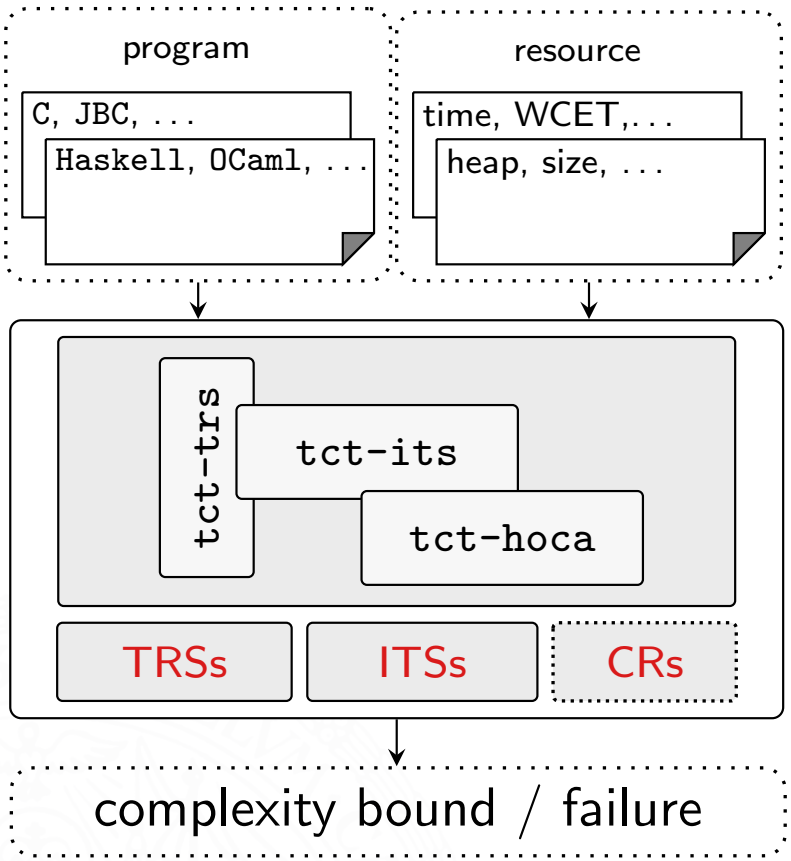



- fully automated resource analysis tool
- competitive for higher-order functional programs, occasionally competitive for object-oriented bytecode programs
- intertwined resource analysis framework

GM and M. Schaper.
From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation.
*IC*. To appear.

# Uniform Resource Analysis

## TcT, AProVE, CiaoPP, COSTA, SACO, ...

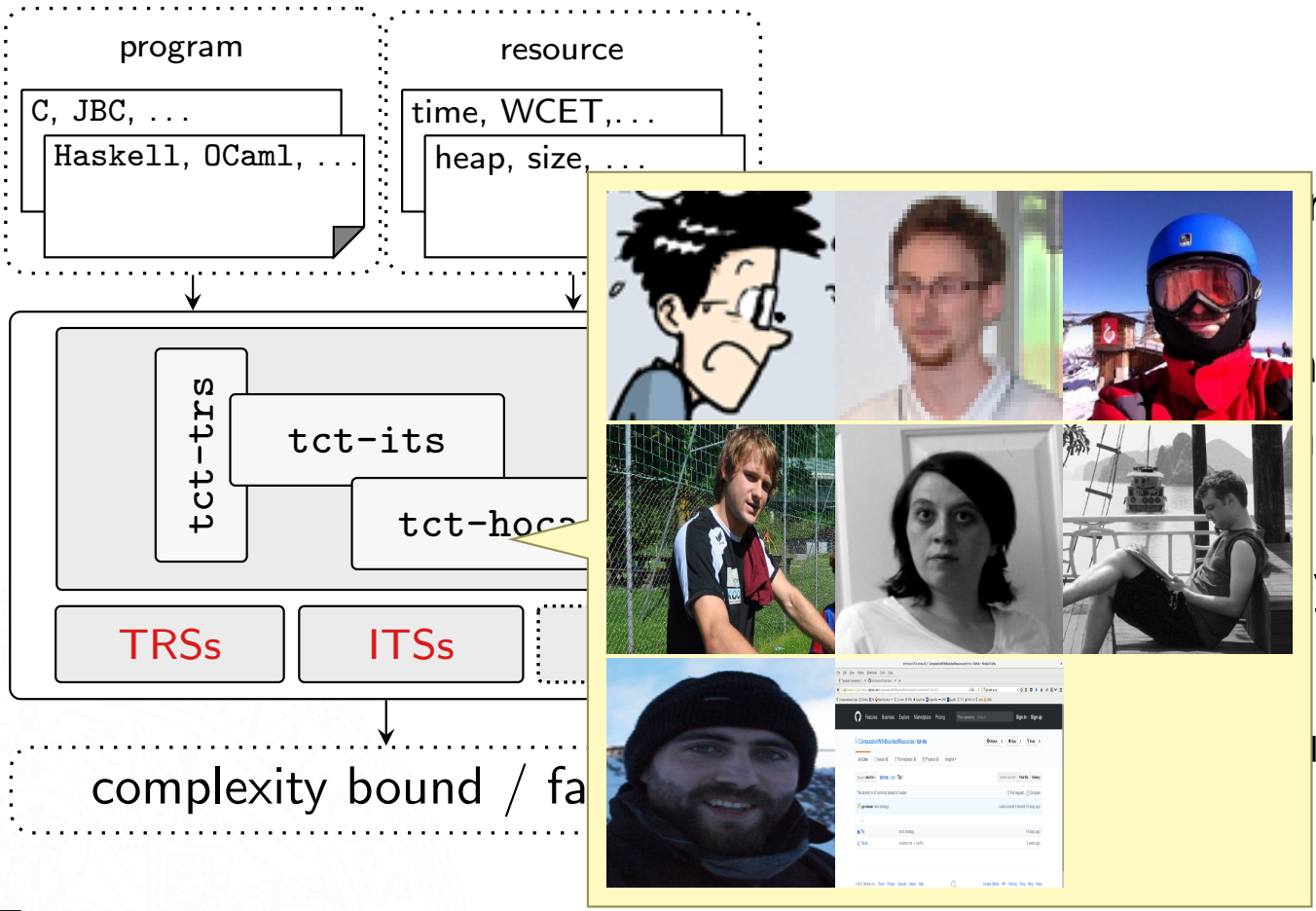



- fully automated resource analysis tool
- competitive for higher-order functional programs, occasionally competitive for object-oriented bytecode programs
- intertwined resource analysis framework

GM and M. Schaper.
From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation.
*IC*. To appear.

# Uniform Resource Analysis

## TcT, AProVE, CiaoPP, COSTA, SACO, ...




resource

igher-order

ms,

petitive for

ytecode

rce analysis

GM and M. Schaper.
From Jinja Bytecode to Term Rewriting: A Complexity Reflecting Transformation.
*IC*. To appear.

# Multivariate Amortised Resource Analysis


J. Hoffmann, K. Aehlig, and M. Hofmann.
Multivariate amortized resource analysis.
*TOPLAS*, 34(3):14, 2012.


J. Hoffmann, A. Das, and S-C. Weng.
Towards automatic resource bound analysis for OCaml.
In *Proc. 44th POPL*, pages 359–373. ACM, 2017.

# Multivariate Amortised Resource Analysis

J. Hoffmann, K. Aehlig, and M. Hofmann.
Multivariate amortized resource analysis.
*TOPLAS*, 34(3):14, 2012.

J. Hoffmann, A. Das, and S-C. Weng.
Towards automatic resource bound analysis for OCaml.
In *Proc. 44th POPL*, pages 359–373. ACM, 2017.

## Results

- state-of-the-art automated resource analysis tool for higher-order functional programs
- type system based, embodying an amortised analysis
- best-case lower bound and worst-case upper bound analysis
- precise, multivariate bounds

# Multivariate Amortised Resource Analysis

J. Hoffmann, K. Aehlig, and M. Hofmann.
Multivariate amortized resource analysis.
*TOPLAS*, 34(3):14, 2012.

J. Hoffmann, A. Das, and S-C. Weng.
Towards automatic resource
In *Proc. 44th POPL*, pages 3

## Results

- state-of-the-art automated r gher-order functional programs
- type system based, embodying an amortised analysis
- best-case lower bound and worst-case upper bound analysis
- precise, multivariate bounds

# Simple and Scalable Static Analysis

M. Sinn, F. Zuleger, and H. Veith.
A simple and scalable static analysis for bound analysis and amortized complexity analysis.
In *Proc. 26th CAV*, volume 8559 of *LNCS*, pages 745–761, 2014.

# Simple and Scalable Static Analysis

M. Sinn, F. Zuleger, and H. Veith.
A simple and scalable static analysis for bound analysis and amortized complexity analysis.
In *Proc. 26th CAV*, volume 8559 of *LNCS*, pages 745–761, 2014.

## Results

- whole program analysis (not composable) for C (LLVM)
- employs vector addition systems with states for program abstraction in conjunction with the synthesis of lexicographic ranking functions
- effective on realistically sized benchmarks and outperforms existing tools (for C or integer transition systems)
- requires the use of additional invariant generations and shape analysis

# Simple and Scalable Static Analysis

M. Sinn, F. Zuleger, and H. Veith.
A simple and scalable static analysis for bound analysis and amortized complexity analysis.
In *Proc. 26th CAV*, volume 8559 of *LNCS*, pages 745–761, 2014.

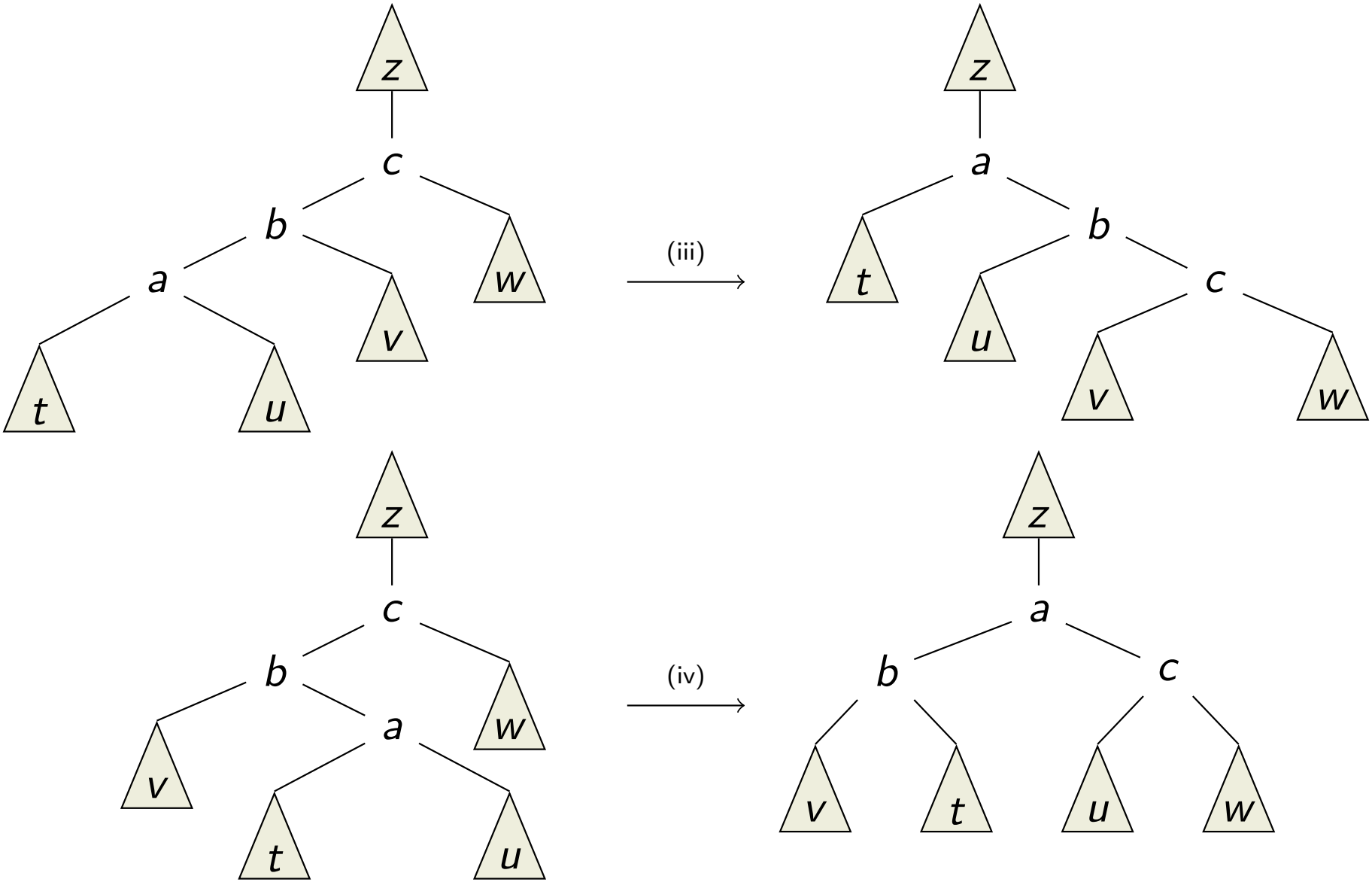
## Results

- whole program analysis (
- employs vector addition s ction in conjunction with the s ions
- effective on realistically sized benchmarks and outperforms existing tools (for C or integer transition systems)
- requires the use of additional invariant generations and shape analysis

## Challenge by Tobias Nipkow (FSCD'16)

*Martin and Georg: this is all very well what you have done with RaML and TCT, but what about real amortised analyses like Tarjan and Sleator's splay trees?*

# Definition (Splaying, interesting cases)



# Amortised Complexity Analysis

## Definition

the potential of a tree is defined as follows

$$\Phi(\text{nil}) := 0$$
$$\Phi(\langle t, a, u \rangle) := \Phi(t) + \log(|t| + |u|) + \Phi(u)$$

where size $|t|$ denotes the number of leaves of $t$

# Amortised Complexity Analysis

## Definition

the potential of a tree is defined as follows

$$\Phi(\text{nil}) := 0$$
$$\Phi(\langle t, a, u \rangle) := \Phi(t) + \log(|t| + |u|) + \Phi(u)$$

where size $|t|$ denotes the number of leaves of $t$

## Definition

the amortised cost of splaying is defined as

$$A(\texttt{splay}\ a\ t) = T(\texttt{splay}\ a\ t) + \Phi(\texttt{splay}\ a\ t) - \Phi(t)$$

# Amortised Complexity Analysis

## Definition

the potential of a tree is defined as follows

$$\Phi(\text{nil}) := 0$$
$$\Phi(\langle t, a, u \rangle) := \Phi(t) + \log(|t| + |u|) + \Phi(u)$$

where size $|t|$ denotes the number of leaves of $t$

## Definition

the amortised cost of splaying is defined as

$$A(\texttt{splay}\ a\ t) = T(\texttt{splay}\ a\ t) + \Phi(\texttt{splay}\ a\ t) - \Phi(t)$$

## Lemma (Sleator, Tarjan)

$A(\texttt{splay}\ a\ t) \leqslant 1 + 3\log(|t|)$

# A Type System for Amortised Resource Analysis

## Preprocessing

- assume a size analysis, resulting in a function types over sized types, for example `splay`: $A \times T_n \to T_n$
- the sized type $T_n$ contains trees of exactly size $n$

# A Type System for Amortised Resource Analysis

## Preprocessing

- assume a size analysis, resulting in a function types over sized types, for example `splay`: $A \times T_n \to T_n$
- the sized type $T_n$ contains trees of exactly size $n$

## Definitions

- let $\Gamma$ denote a typing context and let $\Gamma \supseteq \{T_{n_1}, \ldots, T_{n_k}\}$
- a size polynomial over $\Gamma$ is a multivariate polynomial over $n_1, \ldots, n_k$
- $p \geqslant q$ iff $\forall n_1, \ldots, n_k$

$$\log(p(n_1, \ldots, n_k)) \geqslant \log(q(n_{i_1}, \ldots, n_{i_l}))$$

- an annotated signature decorates function types with size polynomials

$$\texttt{splay}\colon A \times T_n \xrightarrow{n^3+2} T_n$$

## Definition (Type System (selection))

$$\frac{}{\vdash^{1} \mathsf{nil}: T_1} \qquad \frac{n = k + l}{x_1: T_k, x_2: A, x_3: T_l \vdash^{k \cdot l} \langle x_1, x_2, x_3 \rangle: T_n}$$

$$\frac{\Gamma_1 \vdash^{q_1(\Gamma_1)} e_1: D \quad \Gamma_2, x: D \vdash^{q_2(\Gamma_2)} e_2: C \quad q_1 \cdot q_2 \leqslant p}{\Gamma_1, \Gamma_2 \vdash^{p(\Gamma_1, \Gamma_2)} \texttt{let}\ x = e_1\ \texttt{in}\ e_2: C}$$

$$\frac{\Gamma_1 \vdash^{q_1(\Gamma_1)} e_1: C \quad \Gamma_2, x_1: T_k, x_2: A, x_3: T_l \vdash^{q_2(\Gamma_2) \cdot (k+l)} e_2: C \quad \begin{array}{l} m = k + l \\ q_1, q_2 \leqslant p \end{array}}{\Gamma_1, \Gamma_2, x: T_m \vdash^{p(\Gamma_1, \Gamma_2, m)} \texttt{match}\ x\ \texttt{with}\ \mid \mathsf{nil} \rightarrow e_1 \mid \langle x_1, x_2, x_3 \rangle \rightarrow e_2: C}$$

## Definition (Type System (selection))

$$\frac{}{\vdash^{1} \mathsf{nil}: T_1} \qquad \frac{n = k + l}{x_1: T_k, x_2: A, x_3: T_l \vdash^{k \cdot l} \langle x_1, x_2, x_3 \rangle: T_n}$$

$$\frac{\Gamma_1 \vdash^{q_1(\Gamma_1)} e_1: D \quad \Gamma_2, x: D \vdash^{q_2(\Gamma_2)} e_2: C \quad q_1 \cdot q_2 \leqslant p}{\Gamma_1, \Gamma_2 \vdash^{p(\Gamma_1, \Gamma_2)} \texttt{let}\ x = e_1\ \texttt{in}\ e_2: C}$$

$$\frac{\Gamma_1 \vdash^{q_1(\Gamma_1)} e_1: C \quad \Gamma_2, x_1: T_k, x_2: A, x_3: T_l \vdash^{q_2(\Gamma_2) \cdot (k+l)} e_2: C \quad \begin{array}{l} m = k + l \\ q_1, q_2 \leqslant p \end{array}}{\Gamma_1, \Gamma_2, x: T_m \vdash^{p(\Gamma_1, \Gamma_2, m)} \texttt{match}\ x\ \texttt{with}\ \mid \mathsf{nil} \rightarrow e_1 \mid \langle x_1, x_2, x_3 \rangle \rightarrow e_2: C}$$

## Definition

P is called well-typed if $\forall\ f(x_1, \ldots, x_k) = e \in \mathrm{P}$ and $\forall\ C_1 \times \cdots \times C_k \xrightarrow{p} C \in \mathcal{F}(f)$

$$x_1: C_1, \ldots, x_k: C_k \vdash^{p} e: C$$

## Theorem

*Let $\Gamma$ be a typing context and $\sigma$ a substitution consistent with $\Gamma$. Suppose $\sigma \overset{m}{\vdash} e \Rightarrow v$ and $\Gamma \overset{p(\Gamma)}{\vdash} e : C$; we obtain:*

$$\Phi(\sigma, \Gamma) + \log(p(\Gamma)) - \Phi(v) \geqslant m \ .$$

## Proof.

Let $\Xi$ denote the derivation of $\sigma \overset{m}{\vdash} e \Rightarrow v$ and $\Pi$ denote the proof of $\Gamma \overset{p(\Gamma)}{\vdash} e : C$. The theorem is proven by main induction on $|\Pi|$ and side induction on $|\Xi|$ ∎

## Theorem

*Let* $\Gamma$ *be a typing context and* $\sigma$ *a substitution consistent with* $\Gamma$*.*
*Suppose* $\sigma \left|\overset{m}{\vdash}\right. e \Rightarrow v$ *and* $\Gamma \left|\overset{p(\Gamma)}{\vdash}\right. e\colon C$*; we obtain:*

$$\Phi(\sigma, \Gamma) + \log(p(\Gamma)) - \Phi(v) \geqslant m \ .$$

## Proof.

Let $\Xi$ denote the derivation of $\sigma \left|\overset{m}{\vdash}\right. e \Rightarrow v$ and $\Pi$ denote the proof of $\Gamma \left|\overset{p(\Gamma)}{\vdash}\right. e\colon C$. The theorem is proven by main induction on $|\Pi|$ and side induction on $|\Xi|$ ■

## Example

consider the function `splay`: $A \times T_n \to T_n$, defining splaying recursively, then `splay` is well-typed

## Strengths of Uniform Resource Analysis

- modularity and extensiblity
  - complexity problems
  - intermediary languages
- divide and conquer
- applications
- ...

## Strengths of Uniform Resource Analysis

- modularity and extensiblity
  - complexity problems
  - intermediary languages
- divide and conquer
- applications
- ...

## Weaknesses

- extensibility and modularity require abstraction
- abstraction may weaken proving power and may require more work
  - constant amortised analysis (see paper)
  - logarithmic amortised analysis
- ...

## Strengths of Uniform Resource Analysis

- modularity and extensiblity
  - complexity problems
  - intermediary languages
- divide and conquer
- applications
- ...

## Weaknesses

- extensibility and modularity require abstraction
- abstraction may weaken proving power and may require more work
  - constant amortised analysis (see paper)
  - logarithmic amortised analysis
- ...

advice to students: don't listen to advice!

# Thank You for Your Attention!