UNIVERSITÀ di VERONA

Runtime Complexity Analysis of Logically Constrained Rewriting

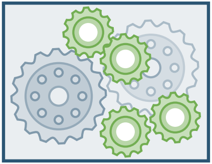
Sarah Winkler and Georg Moser
University of Verona and University of Innsbruck

LOPSTR 2020
7 September 2020

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

⟶



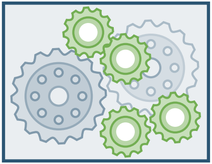
Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$



$\longrightarrow \mathcal{O}(n \log(n))$

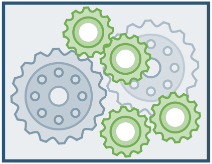
Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$



$\longrightarrow \mathcal{O}(n \log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

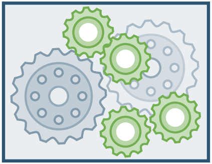
This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$



$\longrightarrow \mathcal{O}(n \log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data stru

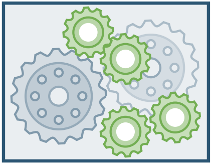
rewrite rules with constraints over SMT-supported theory

This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$  $\longrightarrow \mathcal{O}(n\log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

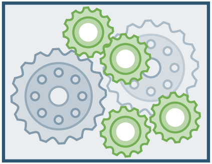
This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):
  - frontends: various programming languages and simplification systems

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 ::  x2 ::  xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$



$\longrightarrow \mathcal{O}(n \log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

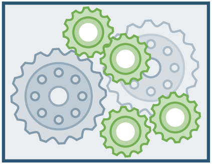
This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):
    - frontends: various programming languages and simplification systems
    - native support for recursion, and arbitrary theories handled by SMT

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 ::  x2 ::  xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

$\longrightarrow$



$\longrightarrow \mathcal{O}(n \log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

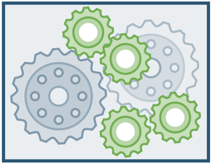
This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):
  - frontends: various programming languages and simplification systems
  - native support for recursion, and arbitrary theories handled by SMT
- fully automatic worst-case runtime analysis, also sub-linear bounds

Motivation: Runtime Analysis

```
mergesort = function
 | [] -> []
 | [x] -> [x]
 | x1 :: x2 :: xs ->
  let (l1,l2) = msplit (x1::x2::xs) in
  merge (mergesort l1, mergesort l2)
...
```

→



⟶ $\mathcal{O}(n \log(n))$

Aims

- fully automatic worst-case runtime analysis
- support for full recursion, common data structures and types

This Talk: Complexity Analysis Framework for LCTRS

- Logically Constrained Rewrite Systems (LCTRS):
  - frontends: various programming languages and simplification systems
  - native support for recursion, and arbitrary theories handled by SMT
- fully automatic worst-case runtime analysis, also sub-linear bounds
- implementation in complexity tool $\mathrm{T_CT}$

Example 1: Integer Transition Systems

$$\mathsf{init}(x,y,z) \to \mathsf{sort}(x,y,z)$$

$$\mathsf{sort}(x,y,z) \to \langle \mathsf{ms}_0(x,u,v), \mathsf{ms}_1(x,u,v), \mathsf{ms}_2(x,u,v), \mathsf{ms}_3(x,u,v)\rangle$$
$$[x \geqslant 2 \wedge u \geqslant 0 \wedge v \geqslant 0 \wedge x+1 \geqslant 2u \wedge 2u \geqslant x \wedge x \geqslant 2v \wedge 2v+1 \geqslant x]$$

$$\mathsf{ms}_0(x,y,z) \to \mathsf{split}(x,y,z) \qquad \mathsf{split}(x,y,z) \to \mathsf{split}(x-2,y,z)\ [x \geqslant 2]$$

$$\mathsf{ms}_1(x,y,z) \to \mathsf{ms}(y,y,z) \qquad \mathsf{merge}(x,y,z) \to \mathsf{merge}(x-1,y,z)\ [x \geqslant 1 \wedge y \geqslant 1]$$

$$\mathsf{ms}_2(x,y,z) \to \mathsf{ms}(z,y,z) \qquad \mathsf{merge}(x,y,z) \to \mathsf{merge}(x,y-1,z)\ [x \geqslant 1 \wedge y \geqslant 1]$$

$$\mathsf{ms}_3(x,y,z) \to \mathsf{merge}(y,z,z)$$

- LCTRSs cover Integer Transition Systems (ITS)

Example 1: Integer Transition Systems

$$\mathsf{init}(x, y, z) \to \mathsf{sort}(x, y, z)$$

$$\mathsf{sort}(x, y, z) \to \langle \mathsf{ms}_0(x, u, v), \mathsf{ms}_1(x, u, v), \mathsf{ms}_2(x, u, v), \mathsf{ms}_3(x, u, v)\rangle$$
$$[x \geqslant 2 \land u \geqslant 0 \land v \geqslant 0 \land x + 1 \geqslant 2u \land 2u \geqslant x \land x \geqslant 2v \land 2v + 1 \geqslant x]$$

$$\mathsf{ms}_0(x, y, z) \to \mathsf{split}(x, y, z) \qquad \mathsf{split}(x, y, z) \to \mathsf{split}(x - 2, y, z)\ [x \geqslant 2]$$

$$\mathsf{ms}_1(x, y, z) \to \mathsf{ms}(y, y, z) \qquad \mathsf{merge}(x, y, z) \to \mathsf{merge}(x - 1, y, z)\ [x \geqslant 1 \land y \geqslant 1]$$

$$\mathsf{ms}_2(x, y, z) \to \mathsf{ms}(z, y, z) \qquad \mathsf{merge}(x, y, z) \to \mathsf{merge}(x, y - 1, z)\ [x \geqslant 1 \land y \geqslant 1]$$

$$\mathsf{ms}_3(x, y, z) \to \mathsf{merge}(y, z, z)$$

- LCTRSs cover Integer Transition Systems (ITS)
- new $\mathsf{T_CT}$ version derives optimal $\mathcal{O}(n \log(n))$ bound

Why?

Example 1: Integer Transition Systems

$$\mathsf{init}(x,y,z) \to \mathsf{sort}(x,y,z)$$

$$\mathsf{sort}(x,y,z) \to \langle \mathsf{ms}_0(x,u,v), \mathsf{ms}_1(x,u,v), \mathsf{ms}_2(x,u,v), \mathsf{ms}_3(x,u,v)\rangle$$
$$[x \geqslant 2 \wedge u \geqslant 0 \wedge v \geqslant 0 \wedge x+1 \geqslant 2u \wedge 2u \geqslant x \wedge x \geqslant 2v \wedge 2v+1 \geqslant x]$$

$$\mathsf{ms}_0(x,y,z) \to \mathsf{split}(x,y,z) \qquad \mathsf{split}(x,y,z) \to \mathsf{split}(x-2,y,z)\ [x \geqslant 2]$$

$$\mathsf{ms}_1(x,y,z) \to \mathsf{ms}(y,y,z) \qquad \mathsf{merge}(x,y,z) \to \mathsf{merge}(x-1,y,z)\ [x \geqslant 1 \wedge y \geqslant 1]$$

$$\mathsf{ms}_2(x,y,z) \to \mathsf{ms}(z,y,z) \qquad \mathsf{merge}(x,y,z) \to \mathsf{merge}(x,y-1,z)\ [x \geqslant 1 \wedge y \geqslant 1]$$

$$\mathsf{ms}_3(x,y,z) \to \mathsf{merge}(y,z,z)$$

- LCTRSs cover Integer Transition Systems (ITS)
- new $\mathsf{T_CT}$ version derives optimal $\mathcal{O}(n \log(n))$ bound
  (CoFloCo, KoAT, PUBS, and previous version of $\mathsf{T_CT}$ at best quadratic)

Example 2: Logic Programs

$$\mathsf{max\_length}(ls, m, l) \to \langle \mathsf{max}(ls, 0, m), \mathsf{len}(ls, l)\rangle$$

$$\mathsf{len}(xs, l) \to \mathsf{len}(t, l - 1)\ [xs \approx h :: t] \qquad \mathsf{len}([], 0) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, n, m)\ [h \leqslant n \wedge xs \approx h :: t] \qquad \mathsf{max}([], m, m) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, h, m)\ [h > n \wedge xs \approx h :: t]$$

- can use approach to analyze (constraint) logic programs

Why?

Example 2: Logic Programs

$$\mathsf{max\_length}(ls, m, l) \to \langle \mathsf{max}(ls, 0, m), \mathsf{len}(ls, l) \rangle$$

$$\mathsf{len}(xs, l) \to \mathsf{len}(t, l-1)\ [xs \approx h :: t] \qquad \mathsf{len}([], 0) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, n, m)\ [h \leqslant n \wedge xs \approx h :: t] \qquad \mathsf{max}([], m, m) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, h, m)\ [h > n \wedge xs \approx h :: t]$$

- can use approach to analyze (constraint) logic programs
- new version of $\mathsf{T_CT}$ can handle LCTRSs corresponding to deterministic Prolog programs over integers and lists

Why?

Example 2: Logic Programs

$$\mathsf{max\_length}(ls, m, l) \to \langle \mathsf{max}(ls, 0, m), \mathsf{len}(ls, l) \rangle$$

$$\mathsf{len}(xs, l) \to \mathsf{len}(t, l - 1)\ [xs \approx h :: t] \qquad \mathsf{len}([], 0) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, n, m)\ [h \leqslant n \wedge xs \approx h :: t] \qquad \mathsf{max}([], m, m) \to \langle\rangle$$

$$\mathsf{max}(xs, n, m) \to \mathsf{max}(t, h, m)\ [h > n \wedge xs \approx h :: t]$$

- can use approach to analyze (constraint) logic programs
- new version of $\mathsf{T_CT}$ can handle LCTRSs corresponding to deterministic Prolog programs over integers and lists
- techniques known to extend to decomposable non-deterministic programs

J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, C. Fuhs.
Symbolic evaluation graphs and term rewriting: a general methodology for analyzing logic programs.
Proc. PPDP 2012, pp. 1–12, 2012.

Example 3: Simplification Systems

```
int foo(int z) {
  int x = 4 * (z | 101);
  return -256 ^x;
}
```

$\overset{*}{\dashrightarrow}$

```
define i32 @foo(i32) #0 {
  %2 = or i32 %0, 101
  %3 = mul nsw i32 4, %2
  %4 = xor i32 -256, %3
  ret i32 %4
}
```

Instcombine →

```
define i32 @foo(i32) #0 {
  %2 = shl i32 %0, 2
  %3 = or i32 %2, 404
  %4 = xor i32 %3, -256
  ret i32 %4
}
```

$\overset{*}{\dashrightarrow}$

```
01011111010101011011010
11111111011101000010
01001110010101100101 0
11111011101110100101 0
```

$$\mathsf{mul}(x, C_1) \to \mathsf{shl}(x, C_2)\ [\ \mathsf{log2}\ (C_1) = C_2 \land \mathsf{isPowerOf2}\ (C_1)\ ]$$

Expression simplifications in compilers

- e.g. in LLVM: multiplications to shifts, reordering bitwise operations, ...

Example 3: Simplification Systems

```
int foo(int z) {
  int x = 4 * (z | 101);
  return -256 ^x;
}
```

⋯* ⋯>

```
define i32 @foo(i32) #0 {
  %2 = or i32 %0, 101
  %3 = mul nsw i32 4, %2
  %4 = xor i32 -256, %3
  ret i32 %4
}
```

Instcombine →

```
define i32 @foo(i32) #0 {
  %2 = shl i32 %0, 2
  %3 = or i32 %2, 404
  %4 = xor i32 %3, -256
  ret i32 %4
}
```

⋯* ⋯>

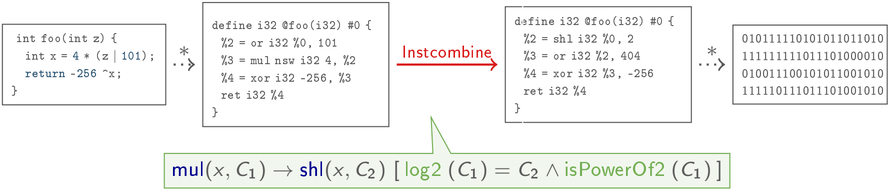
```
01011111010101011011010
1111111110111010000010
01001110010101011001010
111110111011101001010
```

$$\mathsf{mul}(x, C_1) \to \mathsf{shl}(x, C_2)\ [\,\mathsf{log2}\,(C_1) = C_2 \wedge \mathsf{isPowerOf2}\,(C_1)\,]$$

Expression simplifications in compilers

- e.g. in LLVM: multiplications to shifts, reordering bitwise operations, ...
- can be modeled as LCTRS

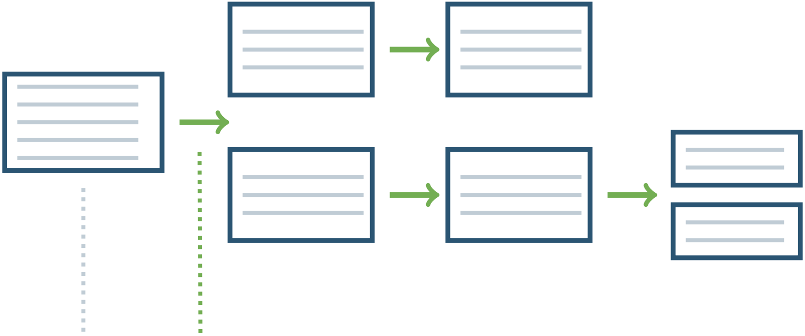
Example 3: Simplification Systems



Expression simplifications in compilers

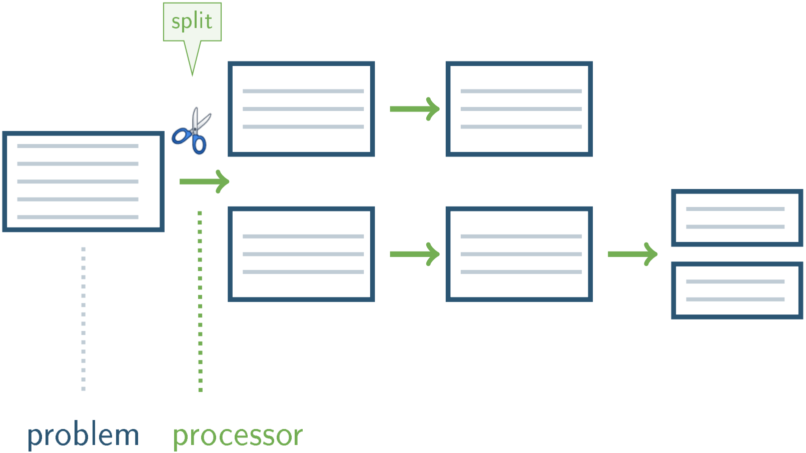
- e.g. in LLVM: multiplications to shifts, reordering bitwise operations, . . .
- can be modeled as LCTRS
- complexity crucial
  (current work is first step: derivational complexity needed)

How? Divide and Conquer

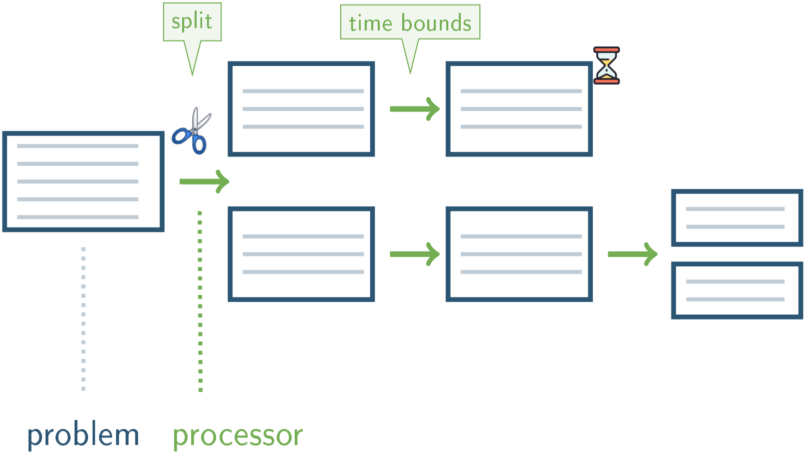


problem processor

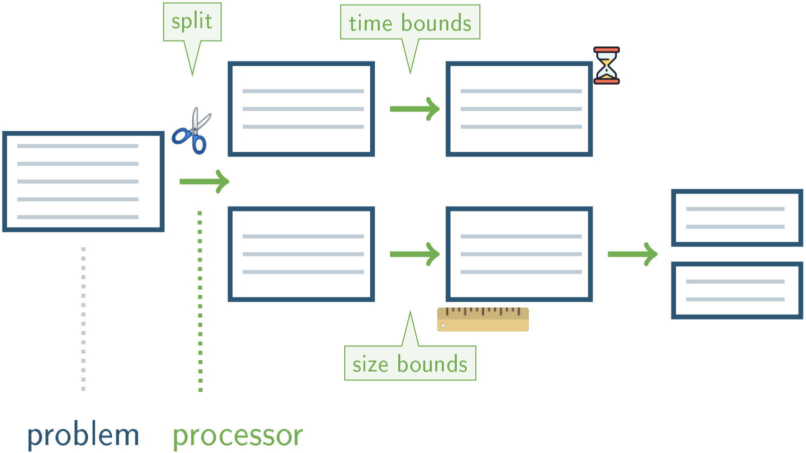
How? Divide and Conquer



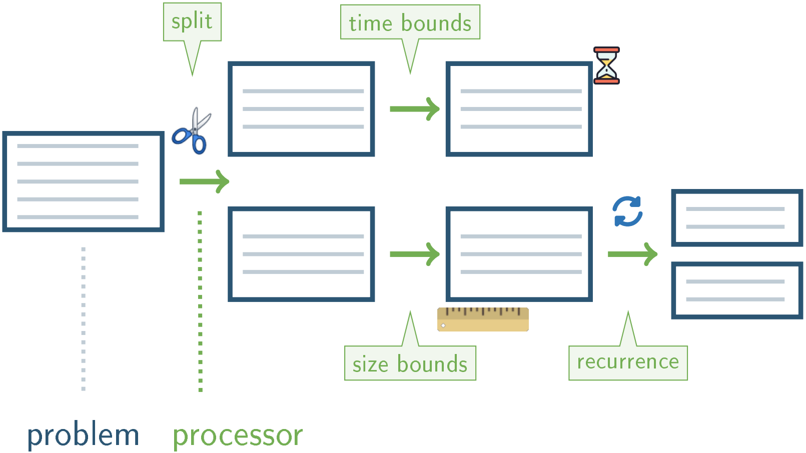
How? Divide and Conquer



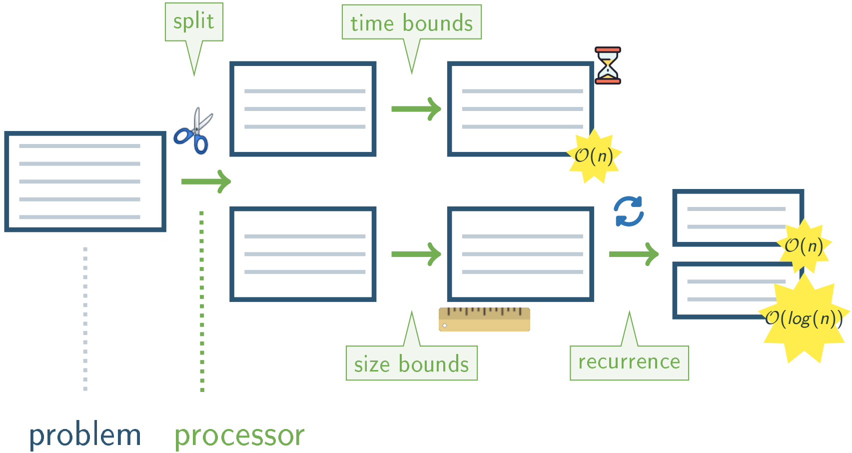
How? Divide and Conquer



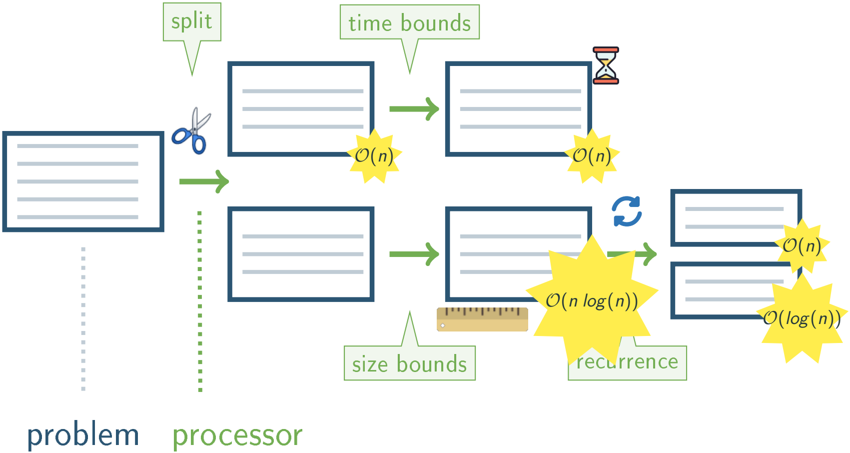
How? Divide and Conquer



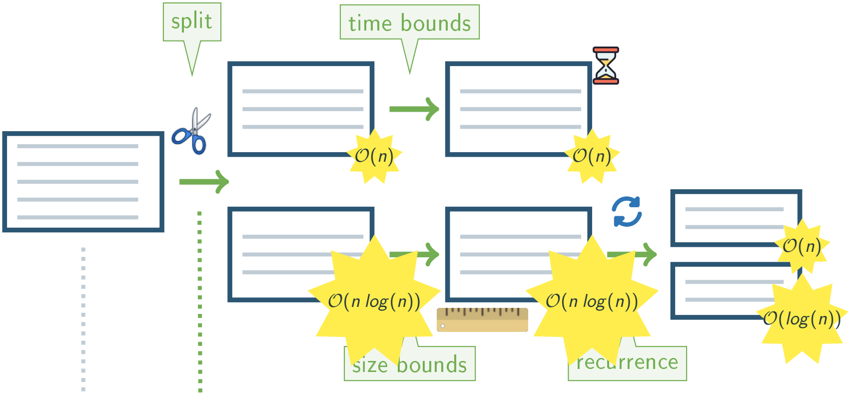
How? Divide and Conquer



How? Divide and Conquer

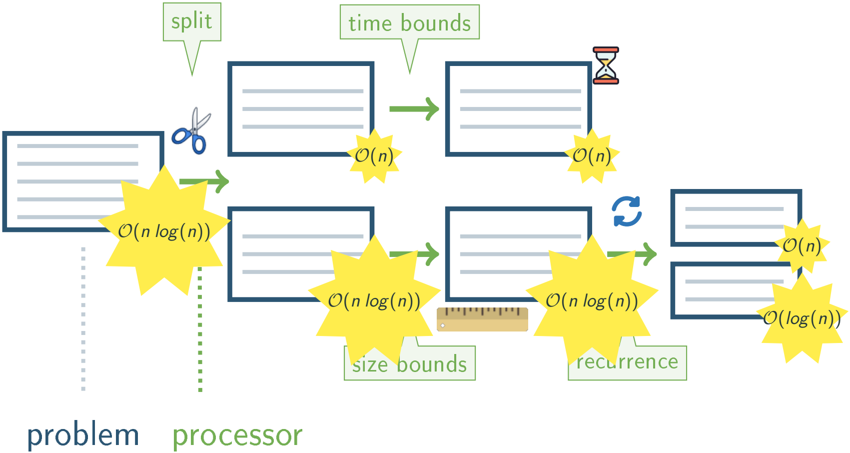


How? Divide and Conquer

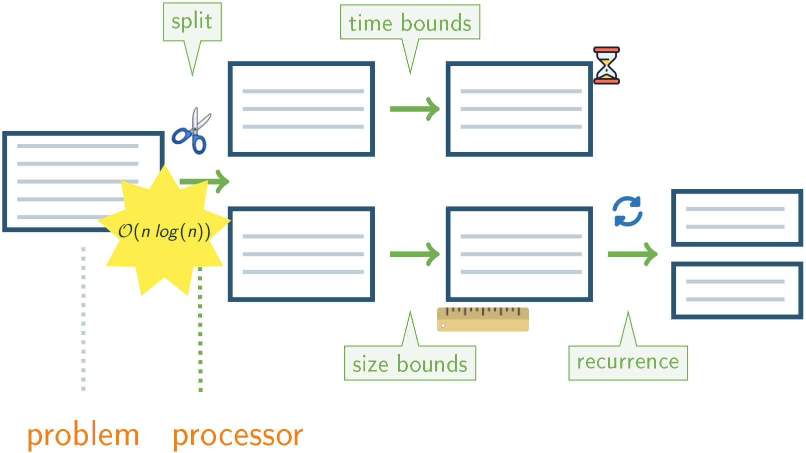


problem processor

How? Divide and Conquer

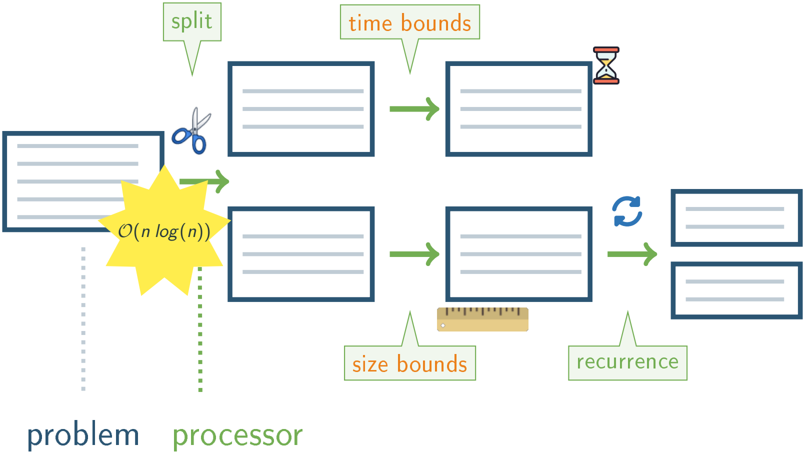


How? Divide and Conquer



M. Avanzini and G. Moser.
A combination framework for complexity.
Inf. Comput., 248:22–55, 2016.

How? Divide and Conquer



M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl.
Analyzing runtime and size complexity of integer programs.
ACM Trans. Program. Lang. Syst., 38(4):13:1–13:50, 2016.

Contents

Logically Constrained Rewrite Systems

- logically constrained rewrite rule

$$\ell \to r\ [c]$$

  - constraint c is term over logic signature (with SMT-decidable theory)
  - terms ℓ, r contain free symbols and logic signature

Logically Constrained Rewrite Systems

- logically constrained rewrite rule

$$\ell \to r \; [c]$$

  - constraint c is term over logic signature (with SMT-decidable theory)
  - terms ℓ, r contain free symbols and logic signature
- LCTRS is set of logically constraint rewrite rules

Logically Constrained Rewrite Systems

- logically constrained rewrite rule

$$\ell \to r\ [c]$$

  - constraint c is term over logic signature (with SMT-decidable theory)
  - terms ℓ, r contain free symbols and logic signature
- LCTRS is set of logically constraint rewrite rules

Example

- $\mathsf{split}(x, y, z) \to \mathsf{split}(x - 2, y, z)\ [x \geqslant 2]$ (integers)

Logically Constrained Rewrite Systems

- logically constrained rewrite rule

$$\ell \to r \ [c]$$

  - constraint c is term over logic signature (with SMT-decidable theory)
  - terms ℓ, r contain free symbols and logic signature
- LCTRS is set of logically constraint rewrite rules

Example

- $\mathsf{split}(x, y, z) \to \mathsf{split}(x - 2, y, z)\ [x \geqslant 2]$ (integers)
- $\mathsf{len}(xs, l) \to \mathsf{len}(t, l - 1)\ [xs \approx h :: t]$ (lists)

Logically Constrained Rewrite Systems

- logically constrained rewrite rule

$$\ell \to r\ [c]$$

  - constraint c is term over logic signature (with SMT-decidable theory)
  - terms ℓ, r contain free symbols and logic signature
- LCTRS is set of logically constraint rewrite rules

Example

- $\mathsf{split}(x, y, z) \to \mathsf{split}(x - 2, y, z)\ [x \geqslant 2]$ (integers)
- $\mathsf{len}(xs, l) \to \mathsf{len}(t, l - 1)\ [xs \approx h :: t]$ (lists)
- $\mathsf{mul}(\mathsf{sub}(y, x), c) \to \mathsf{mul}(\mathsf{sub}(x, y), \mathsf{abs}(c))\ \ [c < \mathbf{0}_8 \wedge \mathsf{isPowerOf2}(\mathsf{abs}(c))]$ (bitvectors)

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r$ $[c]$ is

$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle \; [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r\ [c]$ is

$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle\ [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r
- set of dependency tuples of LCTRS $\mathcal{R}$ is denoted $\mathrm{DT}(\mathcal{R})$

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r$ $[c]$ is

$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle \; [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r
- set of dependency tuples of LCTRS $\mathcal{R}$ is denoted $\mathrm{DT}(\mathcal{R})$

Definition (Dependency graph)

- node set $\mathrm{DT}(\mathcal{R})$ for LCTRS $\mathcal{R}$

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r$ $[c]$ is

$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle \; [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r
- set of dependency tuples of LCTRS $\mathcal{R}$ is denoted $\mathrm{DT}(\mathcal{R})$

Definition (Dependency graph)

- node set $\mathrm{DT}(\mathcal{R})$ for LCTRS $\mathcal{R}$
- edge from $s^{\#} \to \langle \ldots, t^{\#}, \ldots \rangle$ $[\varphi]$ to $u^{\#} \to v^{\#}$ $[\psi]$ if $t^{\#}\sigma \to_{\mathcal{R}}^{*} u^{\#}\tau$

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r\ [c]$ is

$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle\ [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r
- set of dependency tuples of LCTRS $\mathcal{R}$ is denoted $\mathrm{DT}(\mathcal{R})$

Definition (Dependency graph)

- node set $\mathrm{DT}(\mathcal{R})$ for LCTRS $\mathcal{R}$
- edge from $s^{\#} \to \langle \ldots, t^{\#}, \ldots \rangle\ [\varphi]$ to $u^{\#} \to v^{\#}\ [\psi]$ if $t^{\#}\sigma \to_{\mathcal{R}}^{*} u^{\#}\tau$

Example

$$\boxed{\mathsf{init}^{\#}(x) \to \mathsf{f}^{\#}(x)}$$

$$\boxed{\mathsf{f}^{\#}(y) \to \langle \mathsf{f}^{\#}(y-1), \mathsf{g}^{\#}(y) \rangle\ [y \geqslant 0]}$$

$$\boxed{\mathsf{g}^{\#}(z) \to \mathsf{g}^{\#}(z/2)\ [z \geqslant 0]}$$

Definition (Dependency tuples)

- $t^{\#}$ is obtained from term t by marking root symbol by $\#$
- dependency tuple (DT) of $\ell \to r\ [c]$ is

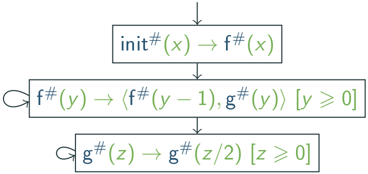
$$\ell^{\#} \to \langle r_1^{\#}, \ldots, r_k^{\#} \rangle\ [c]$$

  where $r_1, \ldots, r_k$ are all recursive calls in r
- set of dependency tuples of LCTRS $\mathcal{R}$ is denoted $\mathsf{DT}(\mathcal{R})$

Definition (Dependency graph)

- node set $\mathsf{DT}(\mathcal{R})$ for LCTRS $\mathcal{R}$
- edge from $s^{\#} \to \langle \ldots, t^{\#}, \ldots \rangle\ [\varphi]$ to $u^{\#} \to v^{\#}\ [\psi]$ if $t^{\#}\sigma \to_{\mathcal{R}}^{*} u^{\#}\tau$

Example



Contents

Bound expressions

$$\text{UB} ::= |x| \mid \text{UB} + \text{UB} \mid \text{UB} \cdot \text{UB} \mid \max(\text{UB}, \text{UB}) \mid \text{UB}^k \mid \log_k(\text{UB}) \mid \omega$$

Bound expressions

$$\mathrm{UB} ::= |x| \mid \mathrm{UB} + \mathrm{UB} \mid \mathrm{UB} \cdot \mathrm{UB} \mid \max(\mathrm{UB}, \mathrm{UB}) \mid \mathrm{UB}^k \mid \log_k(\mathrm{UB}) \mid \omega$$

measure $|x| \in \mathbb{N}$ for all input variables x

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho\colon \ell \to r\ [c] \in \mathsf{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^{\#}(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1}\to \ldots \xrightarrow[\rho]{\sigma_2}\to \ldots \xrightarrow[\rho]{\sigma_m}\to \ldots \qquad (\star)$$

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho \colon \ell \to r\ [c] \in \mathsf{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^{\#}(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1} \to \ldots \xrightarrow[\rho]{\sigma_2} \to \ldots \xrightarrow[\rho]{\sigma_m} \to \ldots \qquad (\star)$$

- time bounds are function $T : \mathsf{DT}(\mathcal{R}) \to \mathsf{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho\colon \ell \to r\,[c] \in \mathsf{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^{\#}(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1} \to \ldots \xrightarrow[\rho]{\sigma_2} \to \ldots$$

expressed in $|x_1|, \ldots, |x_n|$

- time bounds are function $T : \mathsf{DT}(\mathcal{R}) \to \mathsf{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho\colon \ell \to r\ [c] \in \mathsf{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^\#(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1} \to \ldots \xrightarrow[\rho]{\sigma_2} \to \ldots \xrightarrow[\rho]{\sigma_m} \to \ldots \qquad (\star)$$

- time bounds are function $T : \mathsf{DT}(\mathcal{R}) \to \mathsf{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$
- size bounds are function $S : \mathsf{DT}(\mathcal{R}) \times \mathcal{V} \to \mathsf{UB}$ such that $S(\rho, y)$ for $y \in \mathcal{V}\mathsf{ar}(\ell)$ is upper bound on $y\sigma_i$ in $(\star)$

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho\colon \ell \to r\ [c] \in \mathsf{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^\#(x_1, \dots, x_n) \to \dots \xrightarrow[\rho]{\sigma_1}\to \dots \xrightarrow[\rho]{\sigma_2}\to \dots$$

expressed in $|x_1|, \dots, |x_n|$)

- time bounds are function $T : \mathsf{DT}(\mathcal{R}) \to \mathsf{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$
- size bounds are function $S : \mathsf{DT}(\mathcal{R}) \times \mathcal{V} \to \mathsf{UB}$ such that $S(\rho, y)$ for $y \in \mathcal{V}\mathsf{ar}(\ell)$ is upper bound on $y\sigma_i$ in $(\star)$

Bound expressions

$$\mathsf{UB} ::= |x| \mid \mathsf{UB} + \mathsf{UB} \mid \mathsf{UB} \cdot \mathsf{UB} \mid \max(\mathsf{UB}, \mathsf{UB}) \mid \mathsf{UB}^k \mid \log_k(\mathsf{UB}) \mid \omega$$

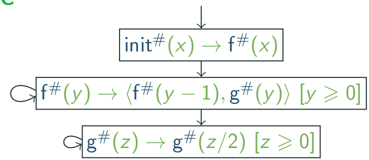
Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho\colon \ell \to r\ [c] \in \mathrm{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathsf{init}^\#(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1}\to \ldots \xrightarrow[\rho]{\sigma_2}\to \ldots \xrightarrow[\rho]{\sigma_m}\to \ldots \qquad (\star)$$

- time bounds are function $T : \mathrm{DT}(\mathcal{R}) \to \mathsf{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$
- size bounds are function $S : \mathrm{DT}(\mathcal{R}) \times \mathcal{V} \to \mathsf{UB}$ such that $S(\rho, y)$ for $y \in \mathcal{V}\mathrm{ar}(\ell)$ is upper bound on $y\sigma_i$ in $(\star)$

Example



$T = 1$

$T = |x|$

$T = |x|^2$

Bound expressions

$$\mathrm{UB} ::= |x| \mid \mathrm{UB} + \mathrm{UB} \mid \mathrm{UB} \cdot \mathrm{UB} \mid \max(\mathrm{UB}, \mathrm{UB}) \mid \mathrm{UB}^k \mid \log_k(\mathrm{UB}) \mid \omega$$

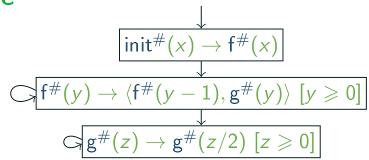
Time bounds and size bounds

for LCTRS $\mathcal{R}$, let $\rho \colon \ell \to r\ [c] \in \mathrm{DT}(\mathcal{R})$ and consider rewrite sequence:

$$\mathrm{init}^{\#}(x_1, \ldots, x_n) \to \ldots \xrightarrow[\rho]{\sigma_1} \to \ldots \xrightarrow[\rho]{\sigma_2} \to \ldots \xrightarrow[\rho]{\sigma_m} \to \ldots \qquad (\star)$$

- time bounds are function $T : \mathrm{DT}(\mathcal{R}) \to \mathrm{UB}$ such that $T(\rho)$ is upper bound on how often ρ is used in $(\star)$
- size bounds are function $S : \mathrm{DT}(\mathcal{R}) \times \mathcal{V} \to \mathrm{UB}$ such that $S(\rho, y)$ for $y \in \mathcal{V}\mathrm{ar}(\ell)$ is upper bound on $y\sigma_i$ in $(\star)$

Example



$T = 1$	$S(x) = \lvert x \rvert$
$T = \lvert x \rvert$	$S(y) = \lvert x \rvert$
$T = \lvert x \rvert^2$	$S(z) = \lvert x \rvert$

Definitions

given LCTRS $\mathcal{R}$,

- complexity problem is $P = (t_0, \mathcal{D}, \mathcal{R})$ for initial term t_0 and DTs $\mathcal{D}$

Processor Framework

Definitions

given LCTRS $\mathcal{R}$,

- complexity problem is $P = (t_0, \mathcal{D}, \mathcal{R})$ for initial term t_0 and DTs $\mathcal{D}$
- judgement $\vdash P\colon (T, S)$ states time bounds T and size bounds S for P

Processor Framework

Definitions

given LCTRS $\mathcal{R}$,

- complexity problem is $P = (t_0, \mathcal{D}, \mathcal{R})$ for initial term t_0 and DTs $\mathcal{D}$
- judgement $\vdash P : (T, S)$ states time bounds T and size bounds S for P
- processor Proc is inference rule on complexity judgements

$$\frac{\vdash P_1 : (T_1, S_1), \ldots, \vdash P_k : (T_k, S_k)}{\vdash P : (T, S)} \quad \text{Proc}$$

Processor Framework

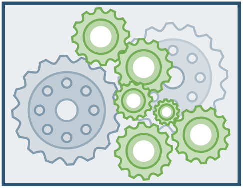
Definitions

given LCTRS $\mathcal{R}$,

- complexity problem is $P = (t_0, \mathcal{D}, \mathcal{R})$ for initial term t_0 and DTs $\mathcal{D}$
- judgement $\vdash P \colon (T, S)$ states time bounds T and size bounds S for P
- processor Proc is inference rule on complexity judgements

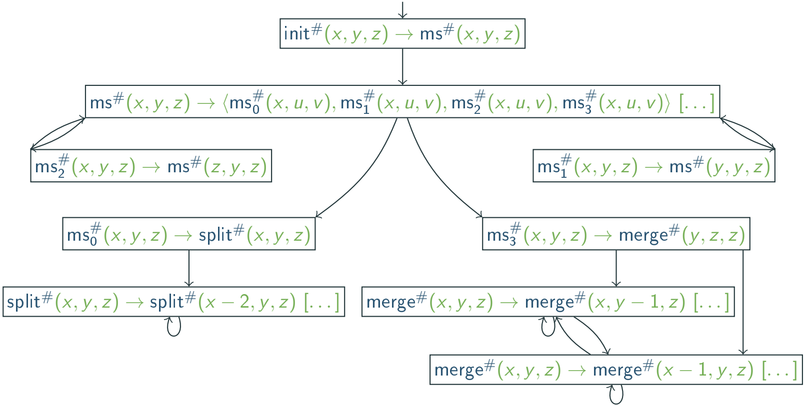
$$\frac{\vdash P_1 \colon (T_1, S_1), \ldots, \vdash P_k \colon (T_k, S_k)}{\vdash P \colon (T, S)} \quad \text{Proc}$$

Processors

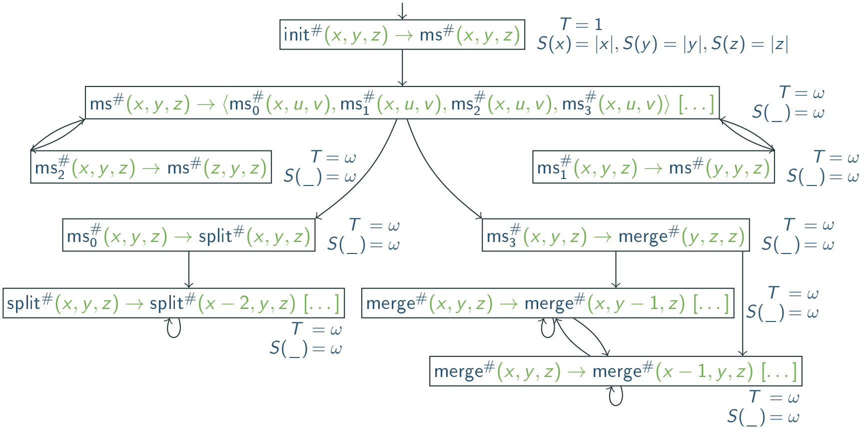


- interpretations
- time bounds
- size bounds
- splitting — new
- recurrence — new
- chaining, simplification

Example (Bounding mergesort)

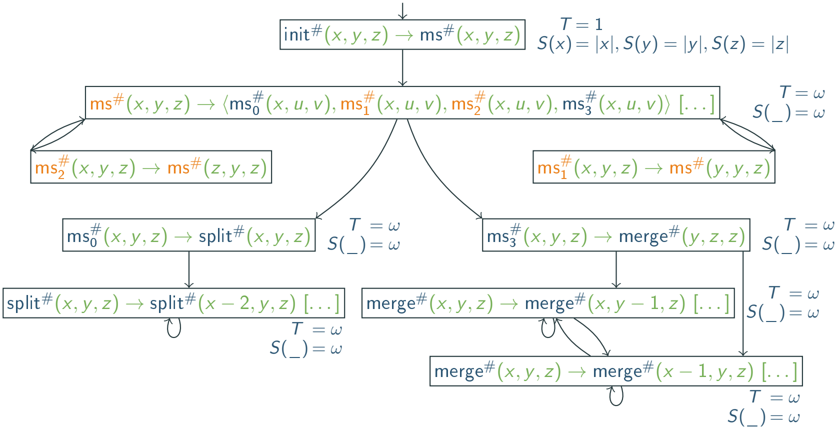


Example (Bounding mergesort)



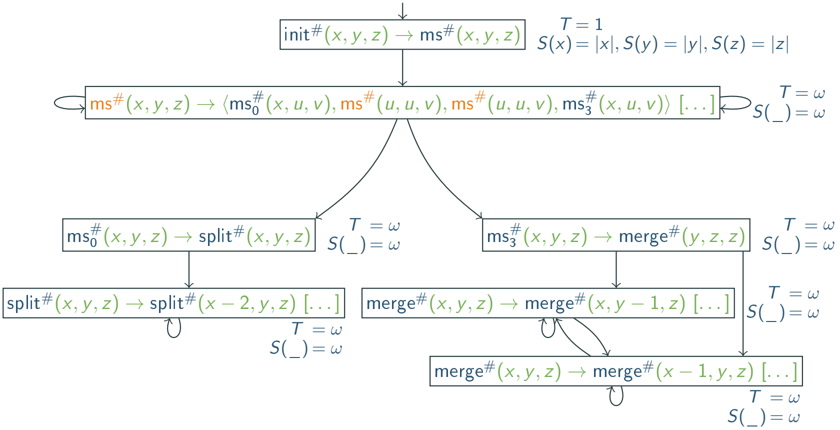
initial problem

Example (Bounding mergesort)



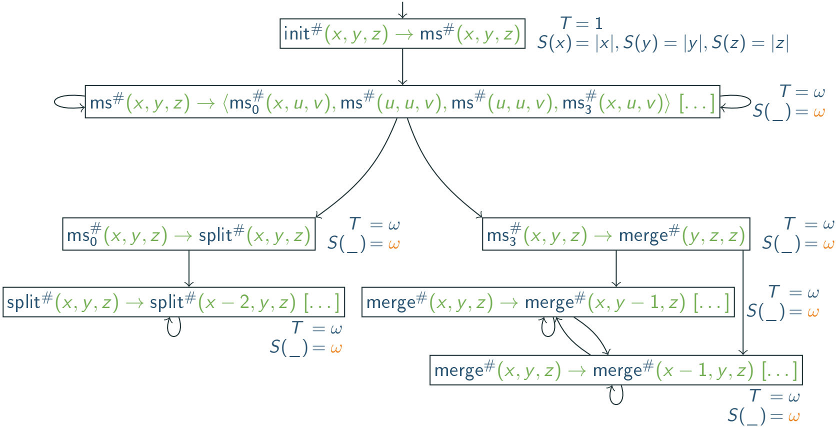
1 Chaining processor

Example (Bounding mergesort)



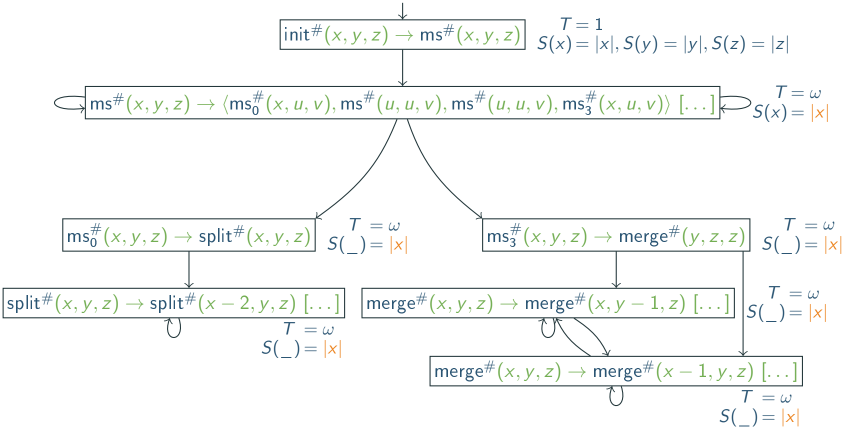
1 Chaining processor

Example (Bounding mergesort)



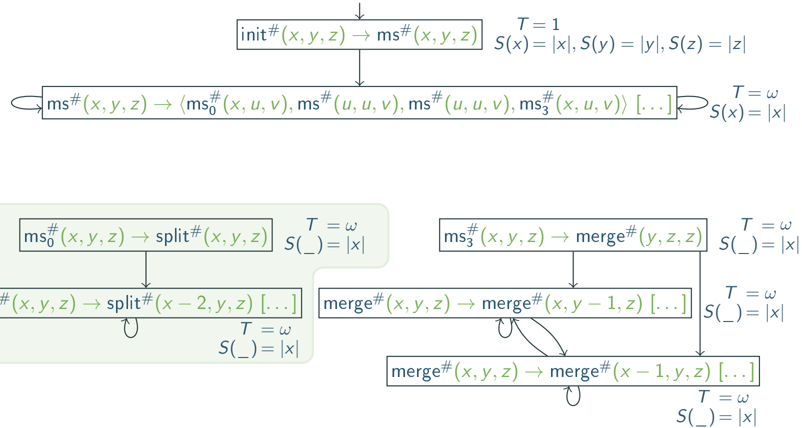
1. Chaining processor
2. Size bounds processor

Example (Bounding mergesort)



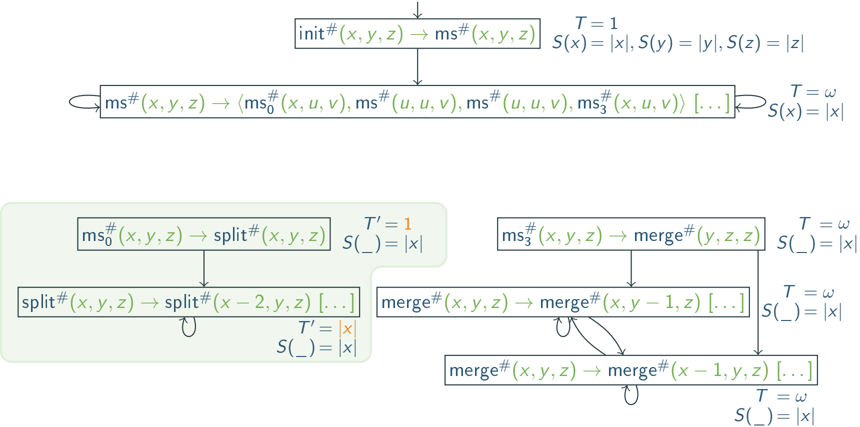
1. Chaining processor
2. Size bounds processor

Example (Bounding mergesort)



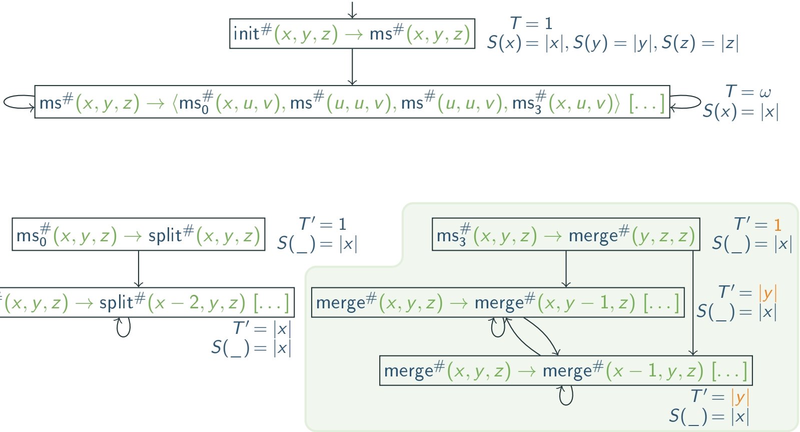
1. Chaining processor
2. Size bounds processor
3. Split processor

Example (Bounding mergesort)



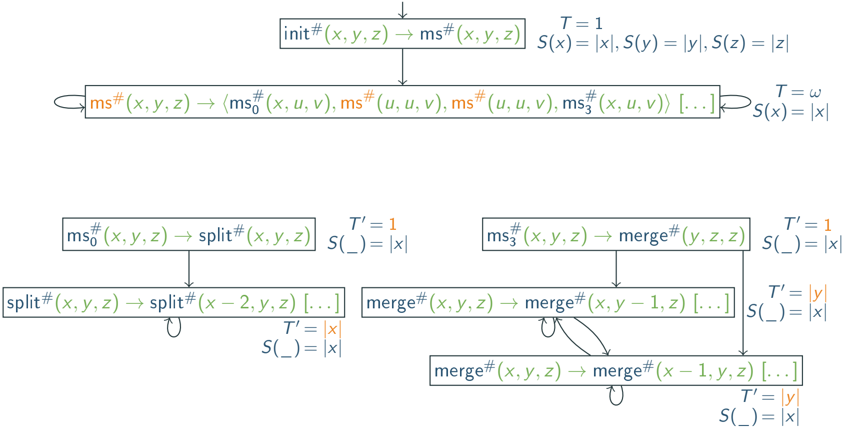
1. Chaining processor
2. Size bounds processor
3. Split processor
4. Interpretation processor

Example (Bounding mergesort)



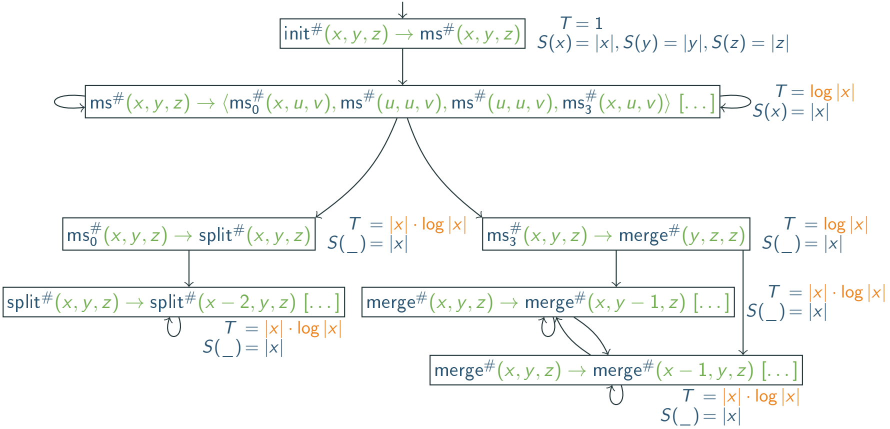
1. Chaining processor
2. Size bounds processor
3. Split processor
4. Interpretation processor
5. Time bounds processor+

Example (Bounding mergesort)



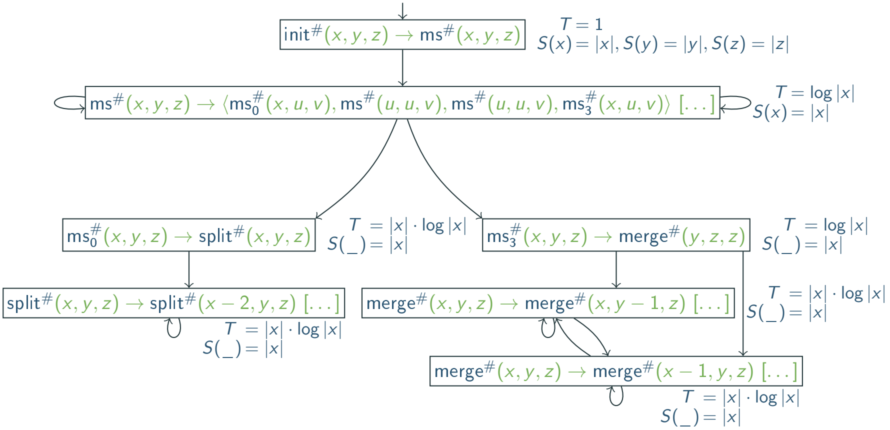
1. Chaining processor
2. Size bounds processor
3. Split processor
4. Interpretation processor
5. Time bounds processor$^+$
6. Recurrence processor

Example (Bounding mergesort)



1. Chaining processor
2. Size bounds processor
3. Split processor
4. Interpretation processor
5. Time bounds processor+
6. Recurrence processor

Example (Bounding mergesort)

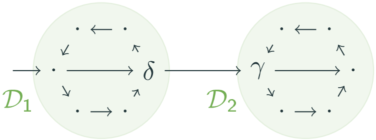


1. Chaining processor
2. Size bounds processor
3. Split processor
4. Interpretation processor
5. Time bounds processor$^+$
6. Recurrence processor

$$\sum T \in \mathcal{O}(|x| \cdot \log |x|)$$

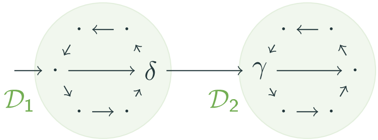
Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

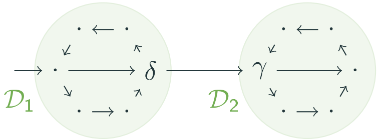


Then the following processor is sound

$$\frac{\vdash P \colon (T, S)}{} \ \text{Split}$$

Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

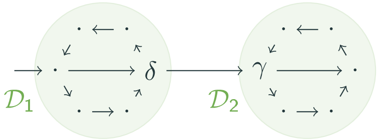


Then the following processor is sound

$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1)}{} \text{ Split}$$

Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

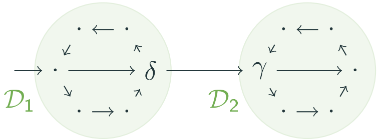


Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

$$\frac{\vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (T, S)}\ \text{Split}$$

Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

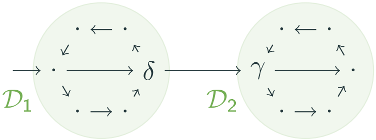


Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \left\{ \qquad \right\}, S)} \quad \text{Split}$$

Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

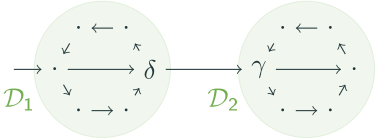


Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ & \end{cases}, S)} \text{ Split}$$

Splitting Processor

Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape

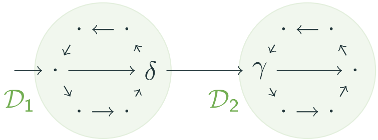


Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta) \cdot T_2(\rho)(S(\gamma, y_1), \ldots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{cases}, S)} \text{ Split}$$

Splitting Processor

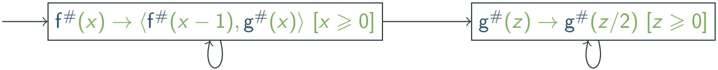
Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

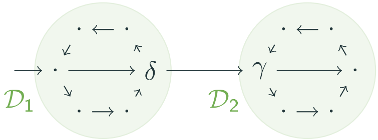
$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta) \cdot T_2(\rho)(S(\gamma, y_1), \ldots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{cases}, S)} \text{ Split}$$

Example



Splitting Processor

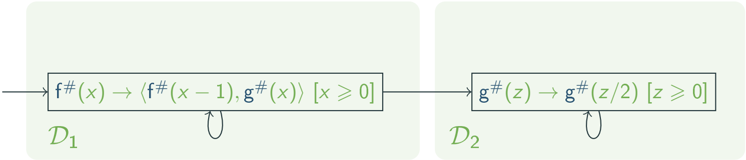
Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Then the following processor is sound, where $\gamma \colon \ell \to r\ [\psi]$

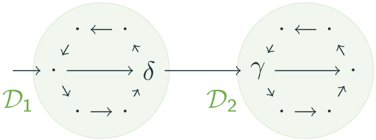
$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta) \cdot T_2(\rho)(S(\gamma, y_1), \dots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{cases}, S)} \text{ Split}$$

Example



Splitting Processor

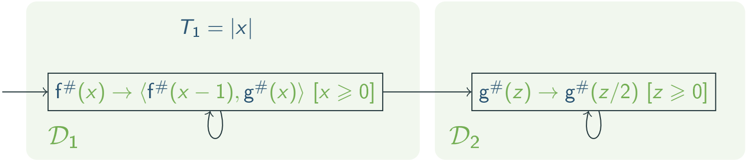
Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Then the following processor is sound, where $\gamma \colon \ell \to r\ [\psi]$

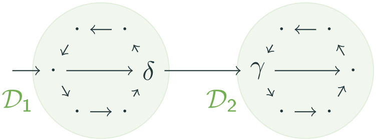
$$\frac{\vdash P\colon (T,S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta)\cdot T_2(\rho)(S(\gamma, y_1), \ldots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{cases}, S)} \ \text{Split}$$

Example



Splitting Processor

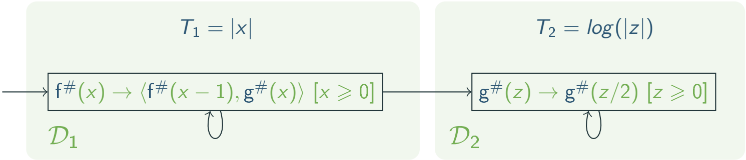
Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Then the following processor is sound, where $\gamma\colon \ell \to r\ [\psi]$

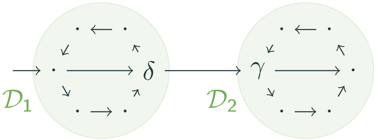
$$\frac{\vdash P\colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R})\colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R})\colon (T_2, S_2)}{\vdash P\colon (\lambda\rho. \begin{cases} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta) \cdot T_2(\rho)(S(\gamma, y_1), \ldots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{cases}, S)} \text{ Split}$$

Example



Splitting Processor

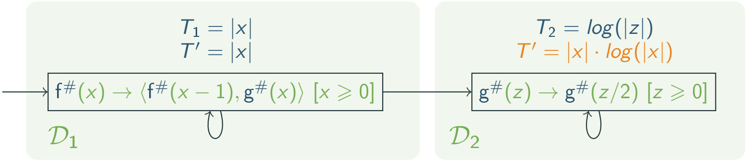
Let $P = (t_0, \mathcal{D}, \mathcal{R})$ have dependency graph of shape



Then the following processor is sound, where $\gamma \colon \ell \to r\ [\psi]$

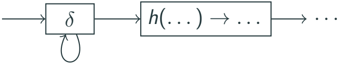
$$\frac{\vdash P \colon (T, S) \quad \vdash (t_0, \mathcal{D}_1, \mathcal{R}) \colon (T_1, S_1) \quad \vdash (\ell, \mathcal{D}_2, \mathcal{R}) \colon (T_2, S_2)}{\vdash P \colon (\lambda\rho. \left\{ \begin{array}{ll} T_1(\rho) & \text{if } \rho \in \mathcal{D}_1 \\ T(\delta) \cdot T_2(\rho)(S(\gamma, y_1), \ldots, S(\gamma, y_k)) & \text{if } \rho \in \mathcal{D}_2 \end{array} \right\}, S)} \ \text{Split}$$

Example



Recurrence Processor

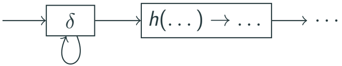
For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



$$\delta : \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t}) \rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



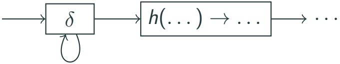
$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) \colon (T, S)}{} \quad \text{Recurrence}$$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



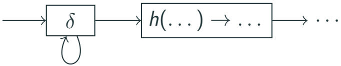
$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) \colon (T, S)}{\vdash P \colon (\lambda\rho. \qquad , S)} \quad \text{Recurrence}$$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



$$\delta : \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

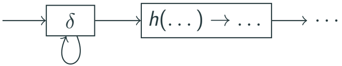
have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) \colon (T, S)}{\vdash P \colon (\lambda\rho. \qquad , S)} \quad \text{Recurrence}$$

if $H > \sum_{\rho \in \mathcal{D} \setminus \{\delta\}} T(\rho)$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \land \vec{x} \geqslant \vec{b}]$$

have following processor:

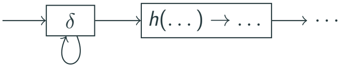
$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) \colon (T, S)}{\vdash P \colon (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

if $H > \sum_{\rho \in \mathcal{D} \setminus \{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x})$, $f(\vec{b}) = 1$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form

form can be generalized



$\delta\colon \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$

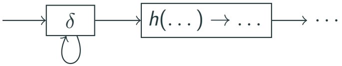
have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R})\colon (T, S)}{\vdash P\colon (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

if $H > \sum_{\rho \in \mathcal{D}\setminus\{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x})$, $f(\vec{b}) = 1$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



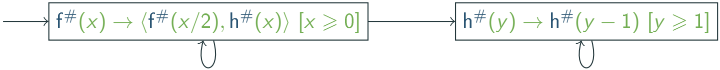
$\delta\colon \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R})\colon (T, S)}{\vdash P\colon (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

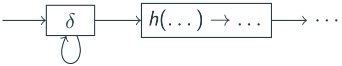
if $H > \sum_{\rho \in \mathcal{D}\setminus\{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x})$, $f(\vec{b}) = 1$

Example



Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



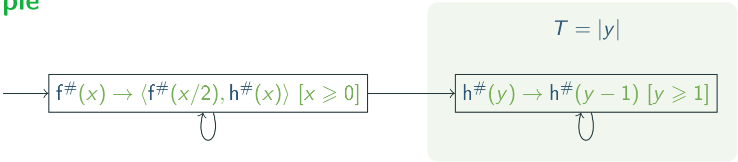
$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) : (T, S)}{\vdash P : (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

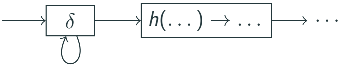
if $H > \sum_{\rho \in \mathcal{D} \setminus \{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x})$, $f(\vec{b}) = 1$

Example



Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



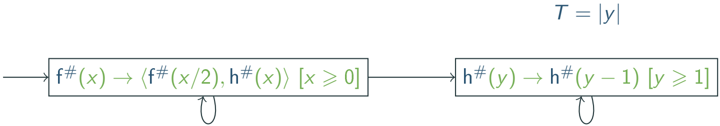
$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t}) \rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) : (T, S)}{\vdash P : (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

if $H > \sum_{\rho \in \mathcal{D} \setminus \{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x})$, $f(\vec{b}) = 1$

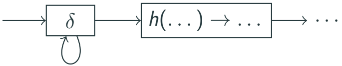
Example



$F = |x| \cdot log(|x|)$ is solution to $f(x) = f(|x|/2) + |x|$, $f(1) = 1$

Recurrence Processor

For complexity problem $P = (f(\vec{x}), \mathcal{D}, \mathcal{R})$ of form



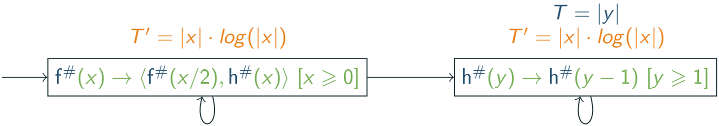
$$\delta: \quad f(\vec{x}) \to \langle f(\vec{r}), h(\vec{t})\rangle \quad [\psi \wedge \vec{x} \geqslant \vec{b}]$$

have following processor:

$$\frac{\vdash (h(\vec{t}), \mathcal{D} \setminus \{\delta\}, \mathcal{R}) \colon (T, S)}{\vdash P \colon (\lambda\rho.\, F(\vec{x}), S)} \quad \text{Recurrence}$$

if $H > \sum_{\rho \in \mathcal{D} \setminus \{\delta\}} T(\rho)$ and F solution to recurrence $f(|\vec{x}|) = f(\vec{r}) + H(\vec{x}),\ f(\vec{b}) = 1$

Example



$F = |x| \cdot log(|x|)$ is solution to $f(x) = f(|x|/2) + |x|,\ f(1) = 1$

Contents

Implementation

- new module of complexity tool $\mathsf{T_CT}$: `tct-lctrs`

Implementation

- new module of complexity tool T_CT: `tct-lctrs`
- integers and lists as background theories (for now), interface Yices/Z3

Implementation

- new module of complexity tool T_CT: `tct-lctrs`
- integers and lists as background theories (for now), interface Yices/Z3
- processors applied according to strategy

`simp`:	unsatisfiable paths, unreachable rules, leaf elimination
`time bounds`:	standard techniques to find ranking functions
`recurrence`:	first solve subproblems, then check whether some recursion pattern according to Master theorem applies

Implementation

- new module of complexity tool $\mathsf{T_CT}$: `tct-lctrs`
- integers and lists as background theories (for now), interface Yices/Z3
- processors applied according to strategy

`simp`:	unsatisfiable paths, unreachable rules, leaf elimination
`time bounds`:	standard techniques to find ranking functions
`recurrence`:	first solve subproblems, then check whether some recursion pattern according to Master theorem applies

- code and results available:

  http://cl-informatik.uibk.ac.at/users/swinkler/lctrs_complexity/

Implementation

- new module of complexity tool TCT: `tct-lctrs`
- integers and lists as background theories (for now), interface Yices/Z3
- processors applied according to strategy

`simp`:	unsatisfiable paths, unreachable rules, leaf elimination
`time bounds`:	standard techniques to find ranking functions
`recurrence`:	first solve subproblems, then check whether some recursion pattern according to Master theorem applies

- code and results available:

  `http://cl-informatik.uibk.ac.at/users/swinkler/lctrs_complexity/`

- ITS benchmarks: optimal, sublinear bounds for several problems where other tools only yield polynomials

Contents

Conclusion

Summary

- first runtime complexity framework for LCTRSs
- advance time/size bound approach by Brockschmidt *et al*,
  combine with complexity framework by Avanzini *et al*, extend to LCTRSs
- additional processors: splitting and recurrence (sublinear bounds)

Conclusion

Summary

- first runtime complexity framework for LCTRSs
- advance time/size bound approach by Brockschmidt *et al*,
  combine with complexity framework by Avanzini *et al*, extend to LCTRSs
- additional processors: splitting and recurrence (sublinear bounds)

Future work

theory:

- more processors: knowledge propagation, narrowing, . . .
- non-innermost rewriting
- analyse derivational complexity

Conclusion

Summary

- first runtime complexity framework for LCTRSs
- advance time/size bound approach by Brockschmidt *et al*, combine with complexity framework by Avanzini *et al*, extend to LCTRSs
- additional processors: splitting and recurrence (sublinear bounds)

Future work

theory:

- more processors: knowledge propagation, narrowing, ...
- non-innermost rewriting
- analyse derivational complexity

applications:

- non-deterministic (constraint) logic programs
- evaluation: ITS benchmarks (+ logic programs, SV-COMP)
- derivational complexity, e.g. for compiler simplification systems