Analysis and Verification of Higher Order Functional Programs: Automata-Theoretic Approach

Ruslán Ledesma Garza
Technische Universität München
ledesmag@in.tum.de

Andrey Rybalchenko
Technische Universität München
rybal@in.tum.de

Automata-theoretic approach

- Only requires reachability and fair termination verifiers
- A standard method for imperative programs

Automata-theoretic approach applied to temporal verification of Ocaml programs with high-order procedures

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

A simple program

```
1+2
```

Evaluation judgement

$$\emptyset \vdash \texttt{1+2} \Rightarrow \texttt{3}$$

Evaluation tree

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow \texttt{1}} \qquad \dfrac{}{\emptyset \vdash \texttt{+} \Rightarrow \texttt{+}}}{\emptyset \vdash \texttt{1+2} \Rightarrow \texttt{3}}$$

Evaluation of 1+2

$$\overline{\qquad\qquad\qquad\qquad}$$
$$\emptyset \vdash \texttt{1+2} \Rightarrow$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow}}{\emptyset \vdash \texttt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2}}}{\emptyset \vdash \mathtt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow}}{\emptyset \vdash \texttt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow \texttt{1}}}{\emptyset \vdash \texttt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\frac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow \texttt{1}} \qquad \dfrac{}{\emptyset \vdash \texttt{+} \Rightarrow}}{\emptyset \vdash \texttt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow \texttt{1}} \qquad \dfrac{}{\emptyset \vdash \texttt{+} \Rightarrow \texttt{+}}}{\emptyset \vdash \texttt{1+2} \Rightarrow}$$

Evaluation of 1+2

$$\dfrac{\dfrac{}{\emptyset \vdash \texttt{2} \Rightarrow \texttt{2}} \qquad \dfrac{}{\emptyset \vdash \texttt{1} \Rightarrow \texttt{1}} \qquad \dfrac{}{\emptyset \vdash \texttt{+} \Rightarrow \texttt{+}}}{\emptyset \vdash \texttt{1+2} \Rightarrow \texttt{3}}$$

A simple monitor

Count additions

`1+2`

Monitored tree

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{\sigma_1^{\uparrow} \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitored tree

Initial state: 0

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow \sigma_2^{\downarrow}} \qquad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow \sigma_3^{\downarrow}} \qquad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{\sigma_1^{\uparrow} \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

Initial state: 0

Current state

$$\dfrac{\dfrac{}{\sigma_2^\uparrow \uparrow \emptyset \vdash \texttt{2} \Rightarrow 2 \downarrow \sigma_2^\downarrow} \quad \dfrac{}{\sigma_3^\uparrow \uparrow \emptyset \vdash \texttt{1} \Rightarrow 1 \downarrow \sigma_3^\downarrow} \quad \dfrac{}{\sigma_4^\uparrow \uparrow \emptyset \vdash \texttt{+} \Rightarrow + \downarrow \sigma_4^\downarrow}}{\sigma_1^\uparrow \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow 3 \downarrow \sigma_1^\downarrow}$$

Monitoring of 1+2

Initial state: 0

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow \sigma_2^{\downarrow}} \qquad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow \sigma_3^{\downarrow}} \qquad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Current state

Monitoring of 1+2

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow \sigma_2^{\downarrow}} \qquad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash 1 \Rightarrow 1 \downarrow \sigma_3^{\downarrow}} \qquad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash + \Rightarrow + \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash 1 + 2 \Rightarrow 3 \downarrow \sigma_1^{\downarrow}}$$

Current state

Monitoring of 1+2

Current state

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow 0} \qquad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \qquad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow 0} \quad \dfrac{}{0 \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow 0} \quad \dfrac{}{0 \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow 0} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \mathtt{1\ +\ 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow 0} \quad \dfrac{}{0 \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow 0} \quad \dfrac{}{0 \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

Current state

$$\frac{\overline{0 \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow 0} \quad \overline{0 \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow 0} \quad \overline{0 \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow 0}}{0 \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

Monitoring of 1+2

Current state

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow 0} \qquad \dfrac{}{0 \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow 0} \qquad \dfrac{}{0 \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow 0}}{0 \uparrow \emptyset \vdash \mathtt{1\ +\ 2} \Rightarrow \mathtt{3} \downarrow 1}$$

Addition counting monitor

$$\mathsf{M}_{+} = (\mathbb{N}, 0, \rho_{+})$$

Addition counting monitor

$$M_+ = (\mathbb{N}, 0, \rho_+)$$

Monitor states

Addition counting monitor

$$\mathsf{M}_+ = (\mathbb{N}, 0, \rho_+)$$

Monitor states

Initial state

Addition counting monitor

$$\mathsf{M}_+ = (\mathbb{N}, 0, \rho_+)$$

Monitor states

Initial state

State transition function

State transition function

$$\mathsf{M}_{+} = (\mathbb{N}, 0, \rho_{+})$$

$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \wedge \\ & \quad \exists \mathcal{E}, e_1, e_2, v : j = \mathcal{E} \vdash e_1 \texttt{ + } e_2 \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

State transition function

$$\mathsf{M}_+ = (\mathbb{N}, 0, \rho_+)$$

$$\rho_+(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \wedge \\ & \quad \exists \mathcal{E}, e_1, e_2, v : j = \mathcal{E} \vdash e_1 + e_2 \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) $\longrightarrow$ State

State transition function

$$M_+ = (\mathbb{N}, 0, \rho_+)$$

$$\rho_+(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \wedge \\ & \quad \exists \mathcal{E}, e_1, e_2, v : j = \mathcal{E} \vdash e_1 + e_2 \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) $\longrightarrow$ State

State transition function

$$\mathsf{M}_+ = (\mathbb{N}, 0, \rho_+)$$

$$\rho_+(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \wedge \\ & \quad \exists \mathcal{E}, e_1, e_2, v : j = \mathcal{E} \vdash e_1 \texttt{ + } e_2 \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) ⟶ State

State transition function

$$\mathsf{M}_{+} = (\mathbb{N}, 0, \rho_{+})$$

$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \wedge \\ & \quad \exists \mathcal{E}, e_1, e_2, v : j = \mathcal{E} \vdash e_1 \;\texttt{+}\; e_2 \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) $\longrightarrow$ State

Monitoring with ρ_+

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{\sigma_1^{\uparrow} \uparrow \emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

$$\rho_+(0, (\emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3}), \uparrow)$$

Monitoring with ρ_{+}

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow \sigma_4^{\downarrow}}}{\sigma_1^{\uparrow} \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow \sigma_1^{\downarrow}}$$

$$\rho_{+}(0, (\emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3}), \uparrow)$$

Monitoring with ρ_+

$$\dfrac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{\sigma_1^{\uparrow} \uparrow \emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

$$\rho_+(0, (\emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3}), \uparrow)$$

Monitoring with ρ_{+}

$$\frac{\dfrac{}{\sigma_2^{\uparrow} \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow \sigma_2^{\downarrow}} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash 1 \Rightarrow 1 \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash + \Rightarrow + \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash 1\ +\ 2 \Rightarrow 3 \downarrow \sigma_1^{\downarrow}}$$

$\rho_{+}(0, (\emptyset \vdash 1\ +\ 2 \Rightarrow 3), \uparrow)$

Tuesday, April 5, 2011

Monitoring with ρ_{+}

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow \sigma_2^{\downarrow}} \qquad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \qquad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

$\rho_{+}(0, (\emptyset \vdash \mathtt{1 + 2} \Rightarrow \mathtt{3}), \uparrow)$

$\rho_{+}(0, (\emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2}), \uparrow)$

Monitoring with ρ_{+}

$$\rho_{+}(0, (\emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2}), \downarrow)$$

$$\dfrac{\dfrac{}{0 \uparrow \emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2} \downarrow 0} \quad \dfrac{}{\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathtt{1} \Rightarrow \mathtt{1} \downarrow \sigma_3^{\downarrow}} \quad \dfrac{}{\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathtt{+} \Rightarrow \mathtt{+} \downarrow \sigma_4^{\downarrow}}}{0 \uparrow \emptyset \vdash \mathtt{1\ +\ 2} \Rightarrow \mathtt{3} \downarrow \sigma_1^{\downarrow}}$$

$$\rho_{+}(0, (\emptyset \vdash \mathtt{1\ +\ 2} \Rightarrow \mathtt{3}), \uparrow)$$

$$\rho_{+}(0, (\emptyset \vdash \mathtt{2} \Rightarrow \mathtt{2}), \uparrow)$$

Monitoring with ρ_+

$$\frac{\overline{0 \uparrow \emptyset \vdash \texttt{2} \Rightarrow \texttt{2} \downarrow 0} \qquad \overline{0 \uparrow \emptyset \vdash \texttt{1} \Rightarrow \texttt{1} \downarrow 0} \qquad \overline{0 \uparrow \emptyset \vdash \texttt{+} \Rightarrow \texttt{+} \downarrow 0}}{0 \uparrow \emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3} \downarrow 1}$$

$$\rho_+(0, (\emptyset \vdash \texttt{1 + 2} \Rightarrow \texttt{3}), \downarrow)$$

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

Product construction

User program

```
let f x =
  if x < 0 then 0 else f (x - 1)
```

Monitor

M

Product construction

User program

```
let f x =
  if x < 0 then 0 else f (x - 1)
```

Monitoring code

```
assert(x > 0)
```

Product construction

User program

```
let f x =
  if x < 0 then 0 else f (x - 1)
```

Monitoring code

```
assert(x > 0)
```

```
let f x = if x < 0 then 0 else assert(x > 0); f (x - 1)
```

Monitor specification

$$S_+ = (0, \texttt{mtrans\_sum})$$

Monitor specification

$$S_{+} = (0, \texttt{mtrans\_sum})$$

Initial state expression

Monitor specification

$$\mathsf{S}_+ = (0, \texttt{mtrans\_sum})$$

Initial state expression

State transformer procedure

State transformer procedure

$$S_+ = (0, \texttt{mtrans\_sum})$$

```
function mtrans_plus counter expression direction =
  match expression, direction with
     | <expr_patt< + $_$ >>, Down ->
        <expr_term< $counter$ + 1 >>
     | _ -> counter
```

State transformer procedure

$$S_+ = (0, \texttt{mtrans\_sum})$$

```
function mtrans_plus counter expression direction =
  match expression, direction with
    | <expr_patt< + $_$ >>, Down ->
       <expr_term< $counter$ + 1 >>
    | _ -> counter
```

(State X Expression X Direction) $\longrightarrow$ Monitoring code

State transformer procedure

$$S_+ = (0, \mathtt{mtrans\_sum})$$

```
function mtrans_plus counter expression direction =
  match expression, direction with
     | <expr_patt< + $_$ >>, Down ->
       <expr_term< $counter$ + 1 >>
     | _ -> counter
```

(State X Expression X Direction) $\longrightarrow$ Monitoring code

Product of 1+2 and M_+

```
1 + 2
```

Product of 1+2 and M_+

```
let x_plus =
  (fun x_11 x_12              -> x_11 + x_12           )
in
let x_1_plus_2                  = x_plus 1 2  in
x_1_plus_2
```

Product of 1+2 and M_+

```
let x_plus =
  (fun x_11 x_12 s_1_pre -> x_11 + x_12, s_1_pre)
in
let x_1_plus_2, s_1_plus_2 = x_plus 1 2 s in
x_1_plus_2, s_1_plus_2 + 1
```

High order program

```
let rec fold_left   f accu l   =
  match l with
    | [] -> accu
    | a::t ->
       let accu'     = f accu a           in
       fold_left   f accu' t
```

High order product

```
let rec fold_left_m f accu l c =
  match l with
    | [] -> accu, c
    | a::t ->
      let accu', c' = f accu a (c + 1) in
      fold_left_m f accu' t c'
```

High order product

```
let rec fold_left_m f accu l c =
  match l with
    | [] -> accu, c
    | a::t ->
      let accu', c' = f accu a (c + 1) in
      fold_left_m f accu' t c'
```

High order product

```
let rec fold_left_m f accu l c =
  match l with
     | [] -> accu, c
     | a::t ->
        let accu', c' = f accu a (c + 1) in
        fold_left_m f accu' t c'
```

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

FunV

- Product construction algorithm
- Reachability checker Dsolve [1]

[1] M. Kawaguchi, P. M. Rondon, and R. Jhala. Type-based data structure verification. In *PLDI*, 2009.

Evaluation

- 600+ LOC
- 62 benchmarks [1–4]

#	Name	Description	Class
1	rev_aux	List reversal helper function.	LinIneq
2	reverse	List reversal function.	LinIneq
3	ins	Ordered insert function	LinIneq
4	clone	Create a pair of copies of a list.	LinIneq
5	tpo	Insert an element in every third position of a list.	LinIneq
6	sec	Remove every third element from a list.	LinIneq
7	tos	Replace each third element of a list.	LinIneq
8	append	List append.	LinIneq
9	split_by	Split list.	LinIneq
10	append	List append.	LinIneq
11	breadth_aux♣	Breadth first tree fold auxiliary function.	LinIneq
12	breadth♣	Breadth first tree fold function.	LinIneq
13	app_tail♡	Apply function to tail of list.	LinIneq
14	compose_list♡	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
15	processfile♠	Open, read, and close one file.	Set
16	processfile♠	Open, read, and close one file.	Set
17	processtwofiles♠	Manipulate two files at the same time.	Set
18	processtwofiles2♠	Manipulate two files at the same time, version 2.	Set
19	mem	Function mem from list.ml.	LinIneq
20	exists♡	Function exists from list.ml.	LinIneq
21	for_all♡	Function for_all from list.ml.	LinIneq
22	rev_map♡	Function rev_map from list.ml.	LinIneq
23	iter♡	Function iter from list.ml.	LinIneq
24	fold_right♡	Function fold_right from list.ml.	LinIneq
25	fold_left♡	Function fold_left from list.ml.	LinIneq
26	map♡	Function map from list.ml.	LinIneq
27	memq	Function memq from list.ml.	LinIneq
28	mem_assoc	Function mem_assoc from list.ml.	LinIneq
29	mem_assq	Function mem_assq from list.ml.	LinIneq
30	remove_assoc	Function remove_assoc from list.ml.	LinIneq
31	find_all♡	Function find_all from list.ml.	LinIneq
32	partition♡	Function partition from list.ml.	LinIneq
33	split	Function split from list.ml.	LinIneq
34	remove_assq	Function remove_assq from list.ml.	LinIneq
35	list_of_tree_aux♣	Auxiliary function for depth first tree fold to list.	LinIneq
36	list_of_tree♣	Function for depth first tree fold to list.	LinIneq
37	list_of_tree2♣	Function for depth first tree fold to list, version 2.	LinIneq
38	tree_of_list♣	Function for constructing a tree from a list.	LinIneq
39	tree_of_list2♣	Function for folding a tree into a list, version 2.	LinIneq
40	tree_flip♣	Function for folding a tree into a list.	LinIneq
41	qsort	Quicksort.	LinIneq
42	ins_sort	Insertion sort.	LinIneq
43	processfile2♠	Open, read, and close one file, version 2.	Set
44	processfile2♠	Open, read, and close one file, version 2.	Set
45	compose_list	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
46	list_of_leaves♣	Left to right list of leaves of a tree.	LinIneq
47	left_path♣	Given a tree, the path from the leftmost leave to the root.	LinIneq,
48	twice	Duplicate every element of a list.	LinIneq
49	add_if_pos	Add a number if it is a positive integer.	LinIneq
50	add_if_pos	Add a number if it is a positive integer.	LinIneq
51	fold_tree♡♣	Pre-order list fold.	LinIneq
52	write_byte♠	Write byte to file.	Set
53	ack	The Ackermann function.	Term
54	chop	Chop the first n elements of a list.	Term
55	dictionary	An algebraic data type recursive manipulation.	Term
56	fold2	Fold a pair of lists.	Term
57	mccarthy91	The McCarthy 91 function.	Term
58	mult	Recursive definition of multiplication.	Term
59	rev_append	Append a list reversed.	Term
60	rev_merge	Merge two lists.	Term
61	simple-rec	A simple recursive function.	Term
62	sum	The sum of the first n naturals.	Term

♡Higher order benchmark.
♣Benchmark on trees.
♠Benchmark on file descriptors.

Figure 8: Experiments.

Evaluation

- 600+ LOC
- 62 benchmarks [1–4]

[1] M. Hofmann. A type system for bounded space and functional in-place update–extended abstract. In *ESOP*, 2000.

[2] M. Hofmann. The strength of non-size increasing computation. In *POPL*, 2002.

[3] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *POPL*, 2003.

[4] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, 2009.

#	Name	Description	Class
1	`rev_aux`	List reversal helper function.	LinIneq
2	`reverse`	List reversal function.	LinIneq
3	`ins`	Ordered insert function	LinIneq
4	`clone`	Create a pair of copies of a list.	LinIneq
5	`tpo`	Insert an element in every third position of a list.	LinIneq
6	`sec`	Remove every third element from a list.	LinIneq
7	`tos`	Replace each third element of a list.	LinIneq
8	`append`	List append.	LinIneq
9	`split_by`	Split list.	LinIneq
10	`append`	List append.	LinIneq
11	`breadth_aux`♣	Breadth first tree fold auxiliary function.	LinIneq
12	`breadth`♣	Breadth first tree fold function.	LinIneq
13	`app_tail`♡	Apply function to tail of list.	LinIneq
14	`compose_list`♡	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
15	`processfile`♠	Open, read, and close one file.	Set
16	`processfile`♠	Open, read, and close one file.	Set
17	`processtwofiles`♠	Manipulate two files at the same time.	Set
18	`processtwofiles2`♠	Manipulate two files at the same time, version 2.	Set
19	`mem`	Function `mem` from list.ml.	LinIneq
20	`exists`♡	Function `exists` from list.ml.	LinIneq
21	`for_all`♡	Function `for_all` from list.ml.	LinIneq
22	`rev_map`♡	Function `rev_map` from list.ml.	LinIneq
23	`iter`♡	Function `iter` from list.ml.	LinIneq
24	`fold_right`♡	Function `fold_right` from list.ml.	LinIneq
25	`fold_left`♡	Function `fold_left` from list.ml.	LinIneq
26	`map`♡	Function `map` from list.ml.	LinIneq
27	`memq`	Function `memq` from list.ml.	LinIneq
28	`mem_assoc`	Function `mem_assoc` from list.ml.	LinIneq
29	`mem_assq`	Function `mem_assq` from list.ml.	LinIneq
30	`remove_assoc`	Function `remove_assoc` from list.ml.	LinIneq
31	`find_all`♡	Function `find_all` from list.ml.	LinIneq
32	`partition`♡	Function `partition` from list.ml.	LinIneq
33	`split`	Function `split` from list.ml.	LinIneq
34	`remove_assq`	Function `remove_assq` from list.ml.	LinIneq
35	`list_of_tree_aux`♣	Auxiliary function for depth first tree fold to list.	LinIneq
36	`list_of_tree`♣	Function for depth first tree fold to list.	LinIneq
37	`list_of_tree2`♣	Function for depth first tree fold to list, version 2.	LinIneq
38	`tree_of_list`♣	Function for constructing a tree from a list.	LinIneq
39	`tree_of_list2`♣	Function for folding a tree into a list, version 2.	LinIneq
40	`tree_flip`♣	Function for folding a tree into a list.	LinIneq
41	`qsort`	Quicksort.	LinIneq
42	`ins_sort`	Insertion sort.	LinIneq
43	`processfile2`♠	Open, read, and close one file, version 2.	Set
44	`processfile2`♠	Open, read, and close one file, version 2.	Set
45	`compose_list`	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
46	`list_of_leaves`♣	Left to right list of leaves of a tree.	LinIneq
47	`left_path`♣	Given a tree, the path from the leftmost leave to the root.	LinIneq,
48	`twice`	Duplicate every element of a list.	LinIneq
49	`add_if_pos`	Add a number if it is a positive integer.	LinIneq
50	`add_if_pos`	Add a number if it is a positive integer.	LinIneq
51	`fold_tree`♡♣	Pre-order list fold.	LinIneq
52	`write_byte`♠	Write byte to file.	Set
53	`ack`	The Ackermann function.	Term
54	`chop`	Chop the first n elements of a list.	Term
55	`dictionary`	An algebraic data type recursive manipulation.	Term
56	`fold2`	Fold a pair of lists.	Term
57	`mccarthy91`	The McCarthy 91 function.	Term
58	`mult`	Recursive definition of multiplication.	Term
59	`rev_append`	Append a list reversed.	Term
60	`rev_merge`	Merge two lists.	Term
61	`simple-rec`	A simple recursive function.	Term
62	`sum`	The sum of the first n naturals.	Term

♡Higher order benchmark.
♣Benchmark on trees.
♠Benchmark on file descriptors.

Figure 8: Experiments.

Evaluation: benchmarks

- Recursion
- Higher-order functions
- Algebraic data types
- Abstract data types

#	Name	Description	Class
1	rev_aux	List reversal helper function.	LinIneq
2	reverse	List reversal function.	LinIneq
3	ins	Ordered insert function	LinIneq
4	clone	Create a pair of copies of a list.	LinIneq
5	tpo	Insert an element in every third position of a list.	LinIneq
6	sec	Remove every third element from a list.	LinIneq
7	tos	Replace each third element of a list.	LinIneq
8	append	List append.	LinIneq
9	split_by	Split list.	LinIneq
10	append	List append.	LinIneq
11	breadth_aux♣	Breadth first tree fold auxiliary function.	LinIneq
12	breadth♣	Breadth first tree fold function.	LinIneq
13	app_tail♡	Apply function to tail of list.	LinIneq
14	compose_list♡	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
15	processfile♠	Open, read, and close one file.	Set
16	processfile♠	Open, read, and close one file.	Set
17	processtwofiles♠	Manipulate two files at the same time.	Set
18	processtwofiles2♠	Manipulate two files at the same time, version 2.	Set
19	mem	Function mem from list.ml.	LinIneq
20	exists♡	Function exists from list.ml.	LinIneq
21	for_all♡	Function for_all from list.ml.	LinIneq
22	rev_map♡	Function rev_map from list.ml.	LinIneq
23	iter♡	Function iter from list.ml.	LinIneq
24	fold_right♡	Function fold_right from list.ml.	LinIneq
25	fold_left♡	Function fold_left from list.ml.	LinIneq
26	map♡	Function map from list.ml.	LinIneq
27	memq	Function memq from list.ml.	LinIneq
28	mem_assoc	Function mem_assoc from list.ml.	LinIneq
29	mem_assq	Function mem_assq from list.ml.	LinIneq
30	remove_assoc	Function remove_assoc from list.ml.	LinIneq
31	find_all♡	Function find_all from list.ml.	LinIneq
32	partition♡	Function partition from list.ml.	LinIneq
33	split	Function split from list.ml.	LinIneq
34	remove_assq	Function remove_assq from list.ml.	LinIneq
35	list_of_tree_aux♣	Auxiliary function for depth first tree fold to list.	LinIneq
36	list_of_tree♣	Function for depth first tree fold to list.	LinIneq
37	list_of_tree2♣	Function for depth first tree fold to list, version 2.	LinIneq
38	tree_of_list♣	Function for constructing a tree from a list.	LinIneq
39	tree_of_list2♣	Function for folding a tree into a list, version 2.	LinIneq
40	tree_flip♣	Function for folding a tree into a list.	LinIneq
41	qsort	Quicksort.	LinIneq
42	ins_sort	Insertion sort.	LinIneq
43	processfile2♠	Open, read, and close one file, version 2.	Set
44	processfile2♠	Open, read, and close one file, version 2.	Set
45	compose_list	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
46	list_of_leaves♣	Left to right list of leaves of a tree.	LinIneq
47	left_path♣	Given a tree, the path from the leftmost leave to the root.	LinIneq,
48	twice	Duplicate every element of a list.	LinIneq
49	add_if_pos	Add a number if it is a positive integer.	LinIneq
50	add_if_pos	Add a number if it is a positive integer.	LinIneq
51	fold_tree♡♣	Pre-order list fold.	LinIneq
52	write_byte♠	Write byte to file.	Set
53	ack	The Ackermann function.	Term
54	chop	Chop the first n elements of a list.	Term
55	dictionary	An algebraic data type recursive manipulation.	Term
56	fold2	Fold a pair of lists.	Term
57	mccarthy91	The McCarthy 91 function.	Term
58	mult	Recursive definition of multiplication.	Term
59	rev_append	Append a list reversed.	Term
60	rev_merge	Merge two lists.	Term
61	simple-rec	A simple recursive function.	Term
62	sum	The sum of the first n naturals.	Term

♡Higher order benchmark.
♣Benchmark on trees.
♠Benchmark on file descriptors.

Figure 8: Experiments.

Evaluation: properties

- Safety and termination
  - Linear inequalities over values and measures
  - Inclusion checks over sets of program values
  - Ranking function/transition invariant checks

#	Name	Description	Class
1	rev_aux	List reversal helper function.	LinIneq
2	reverse	List reversal function.	LinIneq
3	ins	Ordered insert function	LinIneq
4	clone	Create a pair of copies of a list.	LinIneq
5	tpo	Insert an element in every third position of a list.	LinIneq
6	sec	Remove every third element from a list.	LinIneq
7	tos	Replace each third element of a list.	LinIneq
8	append	List append.	LinIneq
9	split_by	Split list.	LinIneq
10	append	List append.	LinIneq
11	breadth_aux♣	Breadth first tree fold auxiliary function.	LinIneq
12	breadth♣	Breadth first tree fold function.	LinIneq
13	app_tail♡	Apply function to tail of list.	LinIneq
14	compose_list♡	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
15	processfile♠	Open, read, and close one file.	Set
16	processfile♠	Open, read, and close one file.	Set
17	processtwofiles♠	Manipulate two files at the same time.	Set
18	processtwofiles2♠	Manipulate two files at the same time, version 2.	Set
19	mem	Function mem from list.ml.	LinIneq
20	exists♡	Function exists from list.ml.	LinIneq
21	for_all♡	Function for_all from list.ml.	LinIneq
22	rev_map♡	Function rev_map from list.ml.	LinIneq
23	iter♡	Function iter from list.ml.	LinIneq
24	fold_right♡	Function fold_right from list.ml.	LinIneq
25	fold_left♡	Function fold_left from list.ml.	LinIneq
26	map♡	Function map from list.ml.	LinIneq
27	memq	Function memq from list.ml.	LinIneq
28	mem_assoc	Function mem_assoc from list.ml.	LinIneq
29	mem_assq	Function mem_assq from list.ml.	LinIneq
30	remove_assoc	Function remove_assoc from list.ml.	LinIneq
31	find_all♡	Function find_all from list.ml.	LinIneq
32	partition♡	Function partition from list.ml.	LinIneq
33	split	Function split from list.ml.	LinIneq
34	remove_assq	Function remove_assq from list.ml.	LinIneq
35	list_of_tree_aux♣	Auxiliary function for depth first tree fold to list.	LinIneq
36	list_of_tree♣	Function for depth first tree fold to list.	LinIneq
37	list_of_tree2♣	Function for depth first tree fold to list, version 2.	LinIneq
38	tree_of_list♣	Function for constructing a tree from a list.	LinIneq
39	tree_of_list2♣	Function for folding a tree into a list, version 2.	LinIneq
40	tree_flip♣	Function for folding a tree into a list.	LinIneq
41	qsort	Quicksort.	LinIneq
42	ins_sort	Insertion sort.	LinIneq
43	processfile2♠	Open, read, and close one file, version 2.	Set
44	processfile2♠	Open, read, and close one file, version 2.	Set
45	compose_list	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
46	list_of_leaves♣	Left to right list of leaves of a tree.	LinIneq
47	left_path♣	Given a tree, the path from the leftmost leave to the root.	LinIneq,
48	twice	Duplicate every element of a list.	LinIneq
49	add_if_pos	Add a number if it is a positive integer.	LinIneq
50	add_if_pos	Add a number if it is a positive integer.	LinIneq
51	fold_tree♡♣	Pre-order list fold.	LinIneq
52	write_byte♠	Write byte to file.	Set
53	ack	The Ackermann function.	Term
54	chop	Chop the first n elements of a list.	Term
55	dictionary	An algebraic data type recursive manipulation.	Term
56	fold2	Fold a pair of lists.	Term
57	mccarthy91	The McCarthy 91 function.	Term
58	mult	Recursive definition of multiplication.	Term
59	rev_append	Append a list reversed.	Term
60	rev_merge	Merge two lists.	Term
61	simple-rec	A simple recursive function.	Term
62	sum	The sum of the first n naturals.	Term

♡Higher order benchmark.
♣Benchmark on trees.
♠Benchmark on file descriptors.

Figure 8: Experiments.

Demo

Conclusion

- Monitoring
- Product construction
- Evaluation