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

Ruslán Ledesma Garza Technische Universität München Iedesmag@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

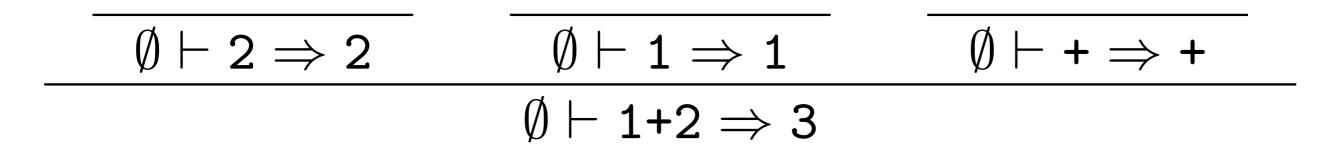
1+2

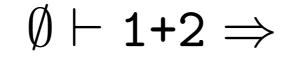
-

Evaluation judgement

$\emptyset \vdash 1+2 \Rightarrow 3$

Evaluation tree





.

 $\emptyset \vdash 2 \Rightarrow$

-

$\emptyset \vdash 1+2 \Rightarrow$

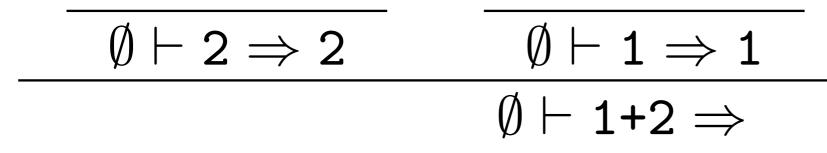
 $\emptyset \vdash 2 \Rightarrow 2$

-

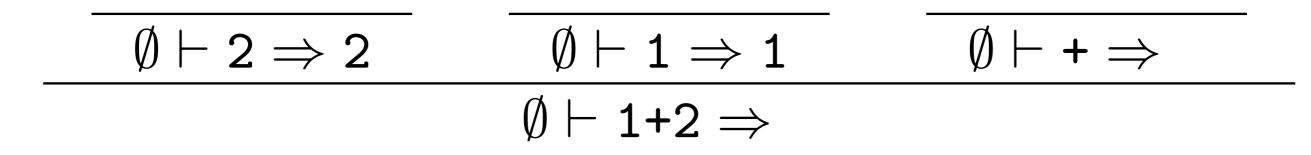
 $\emptyset \vdash 1+2 \Rightarrow$

 $\emptyset \vdash 2 \Rightarrow 2$ $\emptyset \vdash 1 \Rightarrow$ $\emptyset \vdash 1+2 \Rightarrow$

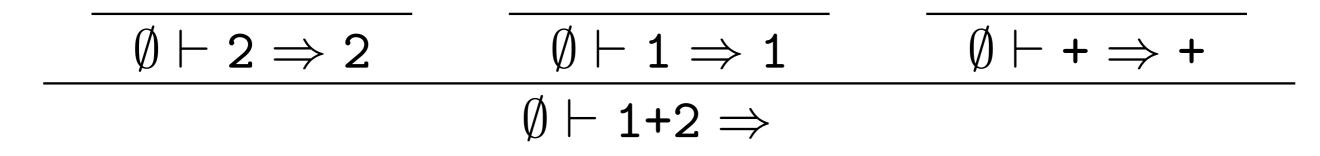
-



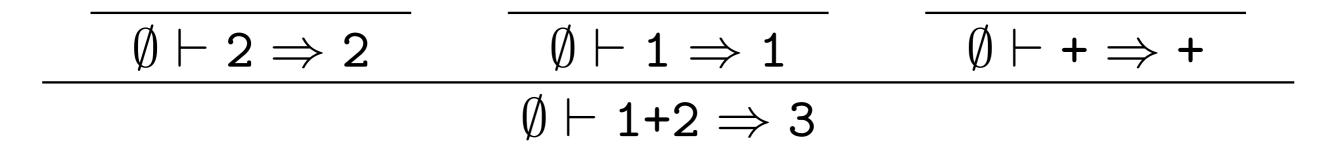
-



-



-



A simple monitor

Count additions

1+2

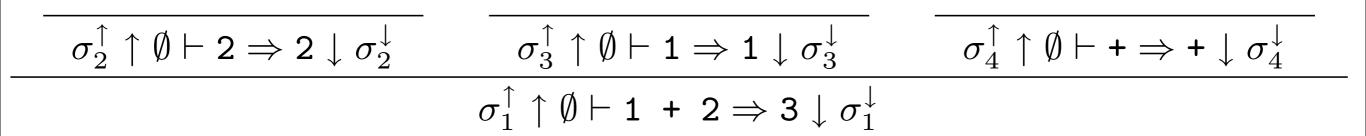
.

Monitored tree

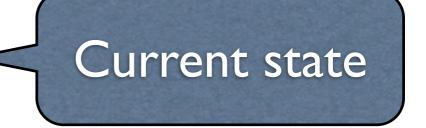
 $\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_3^{\downarrow} \qquad \qquad \sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+} \downarrow \sigma_4^{\downarrow}$ $\sigma_2^\uparrow\uparrow\emptysetdash\mathbf{2}\Rightarrow\mathbf{2}\downarrow\sigma_2^\downarrow$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

Monitored tree

Initial state: 0



Initial state: 0 -

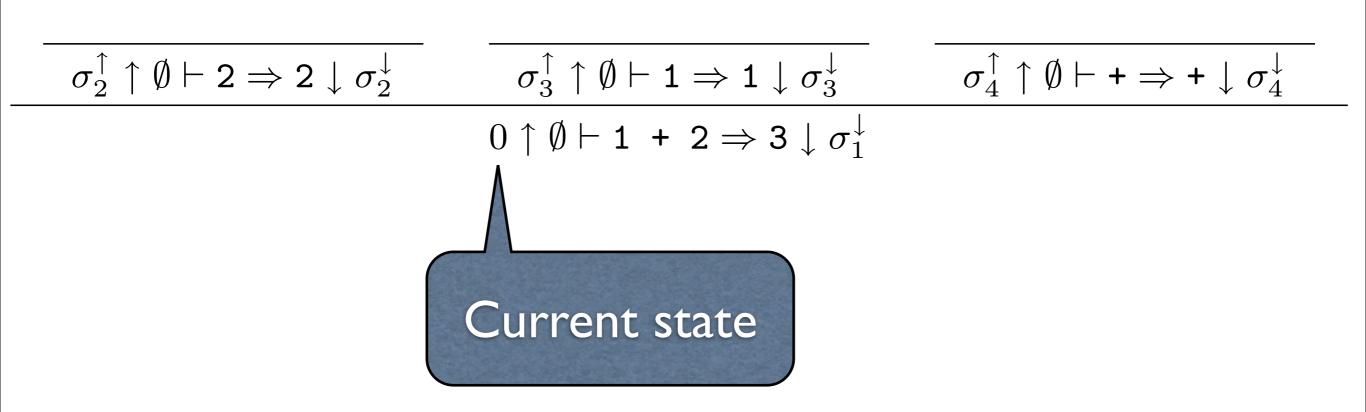


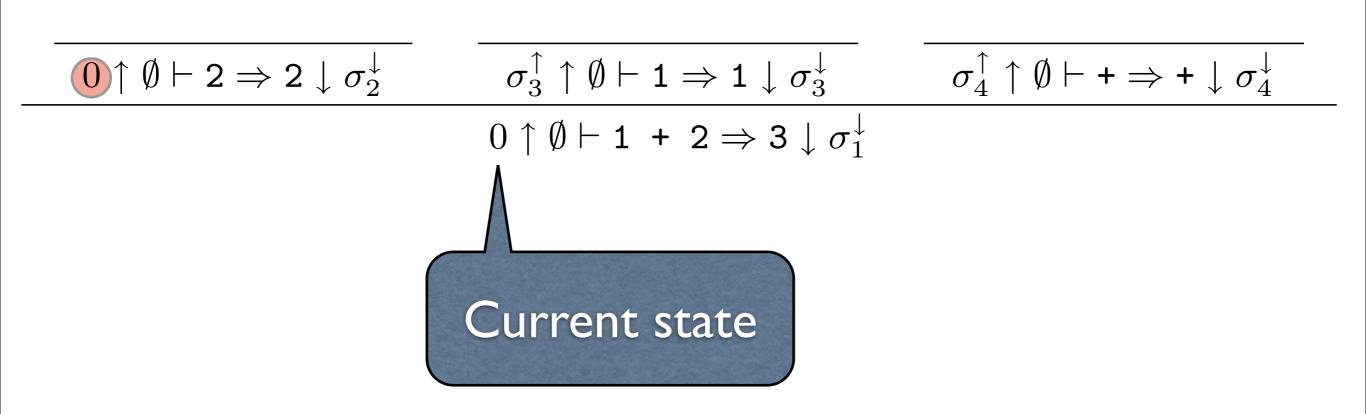
 $\sigma_{3}^{\uparrow} \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_{3}^{\downarrow} \qquad \qquad \sigma_{4}^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+} \downarrow \sigma_{4}^{\downarrow}$ $\sigma_2^{\uparrow} \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow \sigma_2^{\downarrow}$ $\sigma_1^{\uparrow} \uparrow \emptyset \vdash \mathbf{1} + \mathbf{2} \Rightarrow \mathbf{3} \downarrow \sigma_1^{\downarrow}$

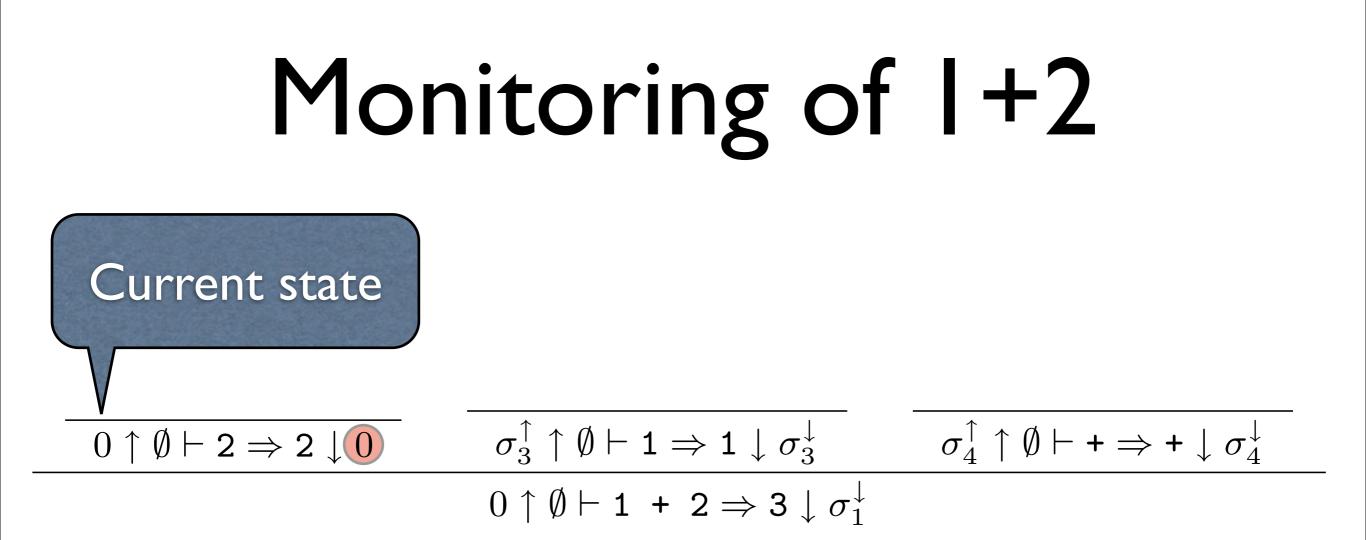
Initial state: 0 -



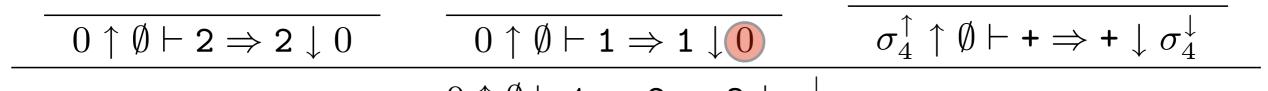
 $\sigma_2^{\uparrow} \uparrow \emptyset \vdash \mathbf{2} \Rightarrow \mathbf{2} \downarrow \sigma_2^{\downarrow}$ $\sigma_3^\uparrow \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_3^\downarrow$ $\sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+}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





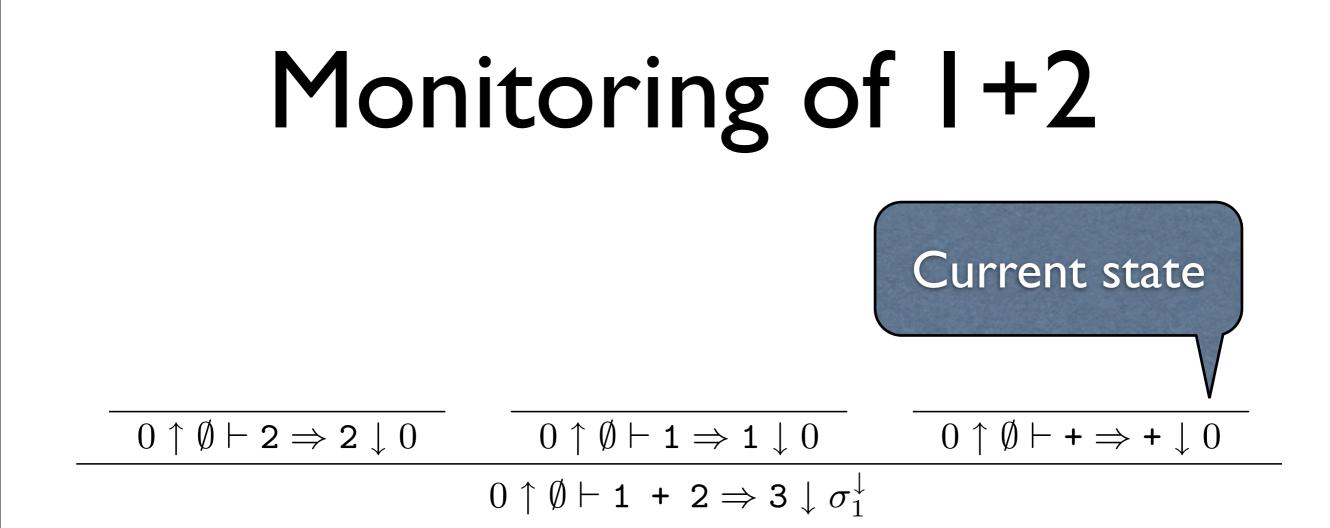


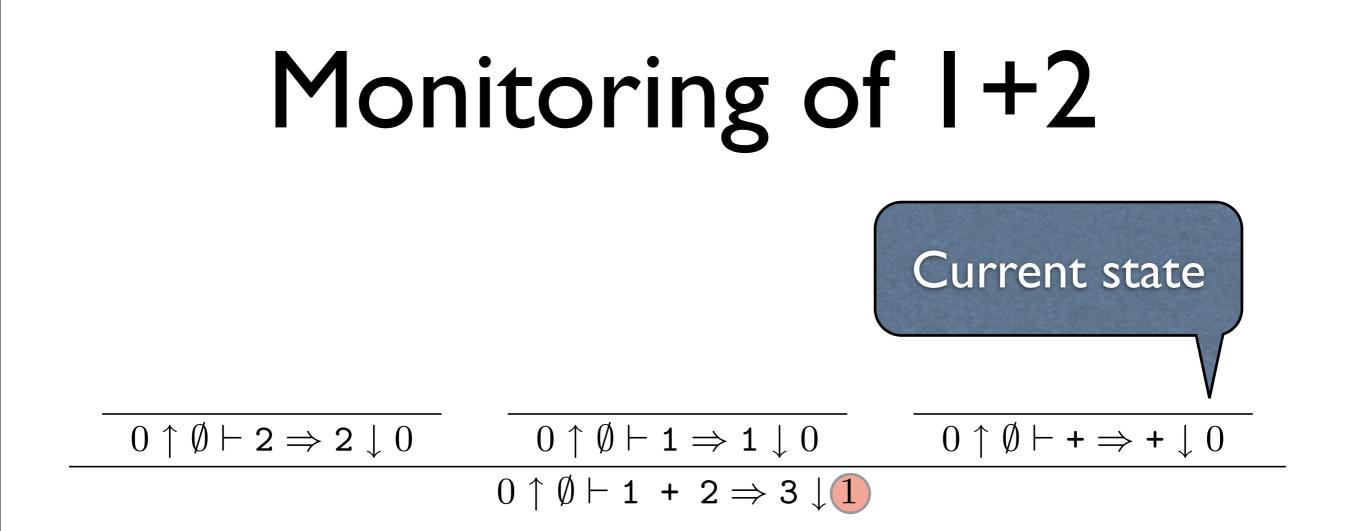
 $\begin{array}{c|c} \hline 0 \uparrow \emptyset \vdash \mathbf{2} \Rightarrow \mathbf{2} \downarrow 0 \\ \hline 0 \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_3^{\downarrow} \\ \hline 0 \uparrow \emptyset \vdash \mathbf{1} + \mathbf{2} \Rightarrow \mathbf{3} \downarrow \sigma_1^{\downarrow} \end{array} \begin{array}{c} \hline \sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+} \downarrow \sigma_4^{\downarrow} \\ \hline \end{array}$



 $0 \uparrow \emptyset \vdash \mathbf{1} + \mathbf{2} \Rightarrow \mathbf{3} \downarrow \sigma_1^{\downarrow}$

 $\begin{array}{c|c}\hline 0 \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow 0 \\ \hline 0 \uparrow \emptyset \vdash 1 \Rightarrow 1 \downarrow 0 \\ \hline 0 \uparrow \emptyset \vdash + \Rightarrow + \downarrow \sigma_4^{\downarrow} \\ \hline 0 \uparrow \emptyset \vdash 1 + 2 \Rightarrow 3 \downarrow \sigma_1^{\downarrow} \end{array}$



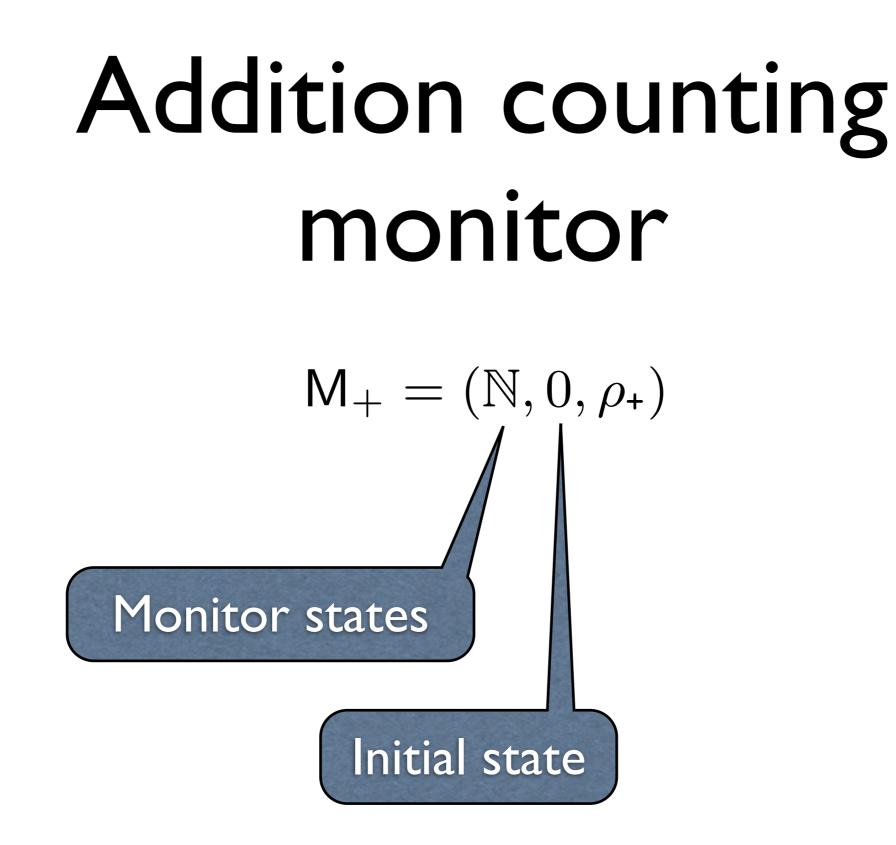


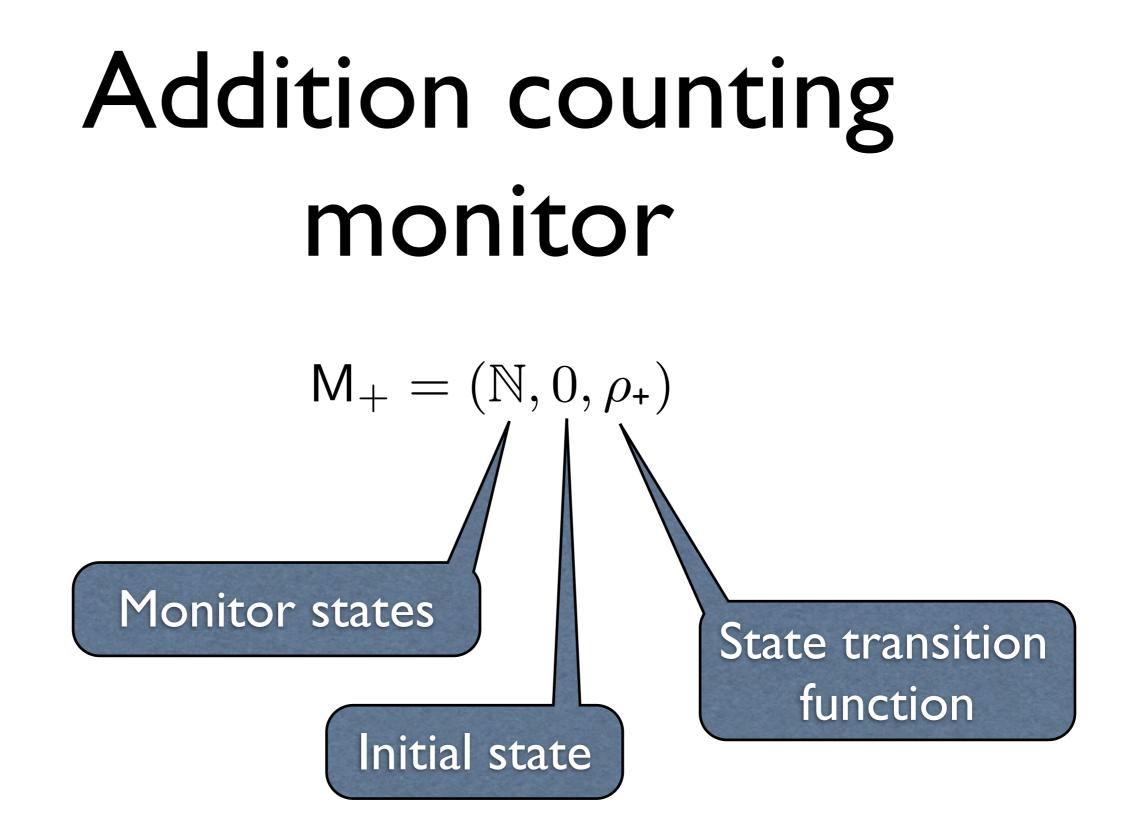
Addition counting monitor

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

Addition counting monitor

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





State transition
function
$$M_{+} = (\mathbb{N}, 0, \rho_{+})$$
$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \land \\ & \exists \mathcal{E}, e_{1}, e_{2}, v : j = \mathcal{E} \vdash e_{1} + e_{2} \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

State transition
function
$$M_{+} = (\mathbb{N}, 0, \rho_{+})$$
$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \land \\ & \exists \mathcal{E}, e_{1}, e_{2}, v : j = \mathcal{E} \vdash e_{1} + e_{2} \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) \rightarrow State

State transition
function

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

$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \land \\ & \exists \mathcal{E}, e_{1}, e_{2}, v : j = \mathcal{E} \vdash e_{1} + e_{2} \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) \rightarrow State

State transition
function

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

$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \land \\ & \exists \mathcal{E}, e_{1}, e_{2}, v : j = \mathcal{E} \vdash e_{1} + e_{2} \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) \rightarrow State

State transition
function
$$M_{+} = (\mathbb{N}, 0, \rho_{+})$$
$$\rho_{+}(\sigma, j, \delta) = \begin{cases} \sigma + 1 & \text{if } \delta = \downarrow \land \\ \exists \mathcal{E}, e_{1}, e_{2}, v : j = \mathcal{E} \vdash e_{1} + e_{2} \Rightarrow v \\ \sigma & \text{otherwise} \end{cases}$$

(State X Judgement X Direction) \rightarrow State

$$\begin{array}{c} \hline \sigma_{2}^{\uparrow} \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow \sigma_{2}^{\downarrow} & \hline \sigma_{3}^{\uparrow} \uparrow \emptyset \vdash 1 \Rightarrow 1 \downarrow \sigma_{3}^{\downarrow} & \hline \sigma_{4}^{\uparrow} \uparrow \emptyset \vdash + \Rightarrow + \downarrow \sigma_{4}^{\downarrow} \\ \\ \hline \sigma_{1}^{\uparrow} \uparrow \emptyset \vdash 1 + 2 \Rightarrow 3 \downarrow \sigma_{1}^{\downarrow} & \end{array}$$

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

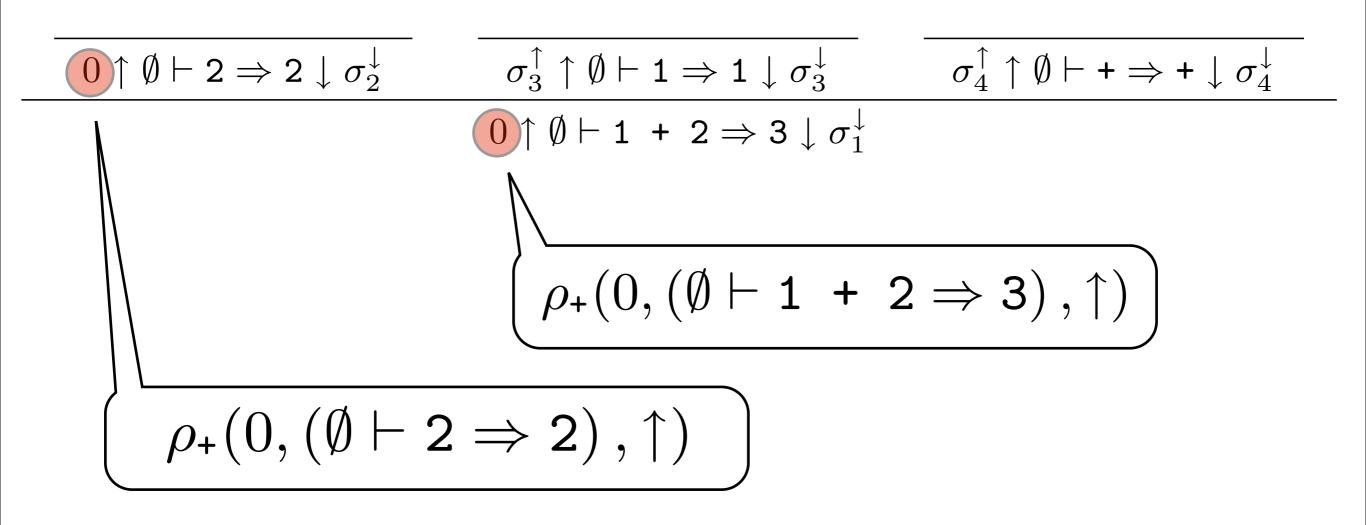
 $\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_3^{\downarrow} \qquad \qquad \sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+} \downarrow \sigma_4^{\downarrow}$ $\sigma_2^\uparrow \uparrow \emptyset \vdash 2 \Rightarrow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

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

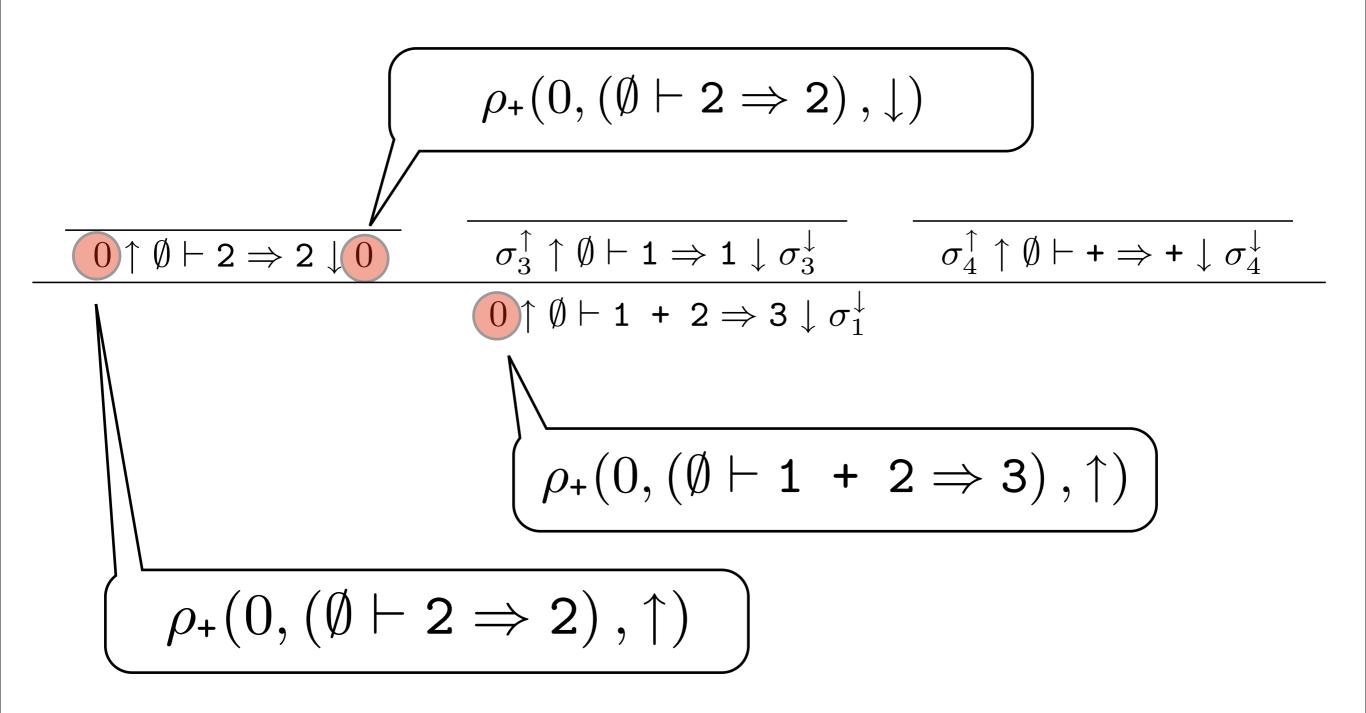
 $\sigma_3^{\uparrow} \uparrow \emptyset \vdash \mathbf{1} \Rightarrow \mathbf{1} \downarrow \sigma_3^{\downarrow} \qquad \qquad \sigma_4^{\uparrow} \uparrow \emptyset \vdash \mathbf{+} \Rightarrow \mathbf{+} \downarrow \sigma_4^{\downarrow}$ $\sigma_2^{\uparrow} \uparrow \emptyset \vdash 2 \Rightarrow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

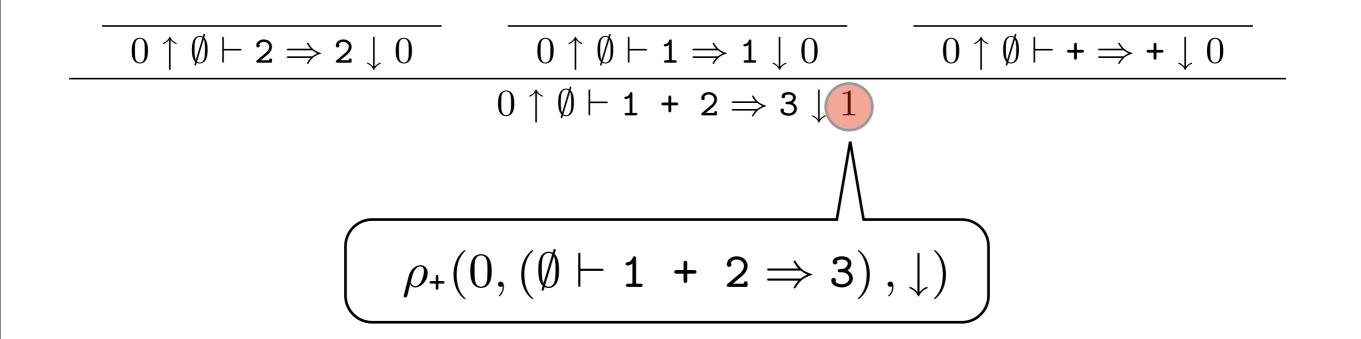
$$\left(\rho_{+}(0, (\emptyset \vdash 1 + 2 \Rightarrow 3), \uparrow)\right)$$

$$\begin{array}{c|c} \hline \sigma_{2}^{\uparrow} \uparrow \emptyset \vdash 2 \Rightarrow 2 \downarrow \sigma_{2}^{\downarrow} & \hline \sigma_{3}^{\uparrow} \uparrow \emptyset \vdash 1 \Rightarrow 1 \downarrow \sigma_{3}^{\downarrow} & \hline \sigma_{4}^{\uparrow} \uparrow \emptyset \vdash + \Rightarrow + \downarrow \sigma_{4}^{\downarrow} \\ \hline 0 \uparrow \emptyset \vdash 1 + 2 \Rightarrow 3 \downarrow \sigma_{1}^{\downarrow} \\ \hline \rho_{+}(0, (\emptyset \vdash 1 + 2 \Rightarrow 3), \uparrow) \\ \end{array}$$









Our work

- Monitoring for evaluation trees
- Product construction
- Evaluation

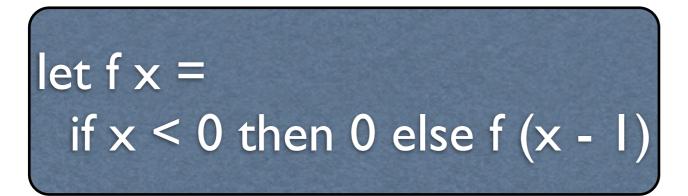
Our work

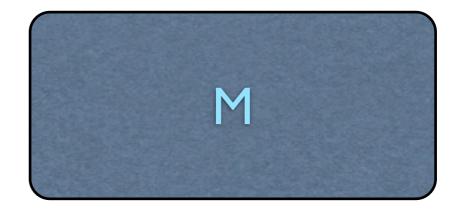
- Monitoring for evaluation trees
- Product construction
- Evaluation

Product construction

User program

Monitor



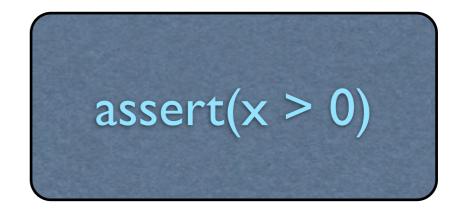


Product construction

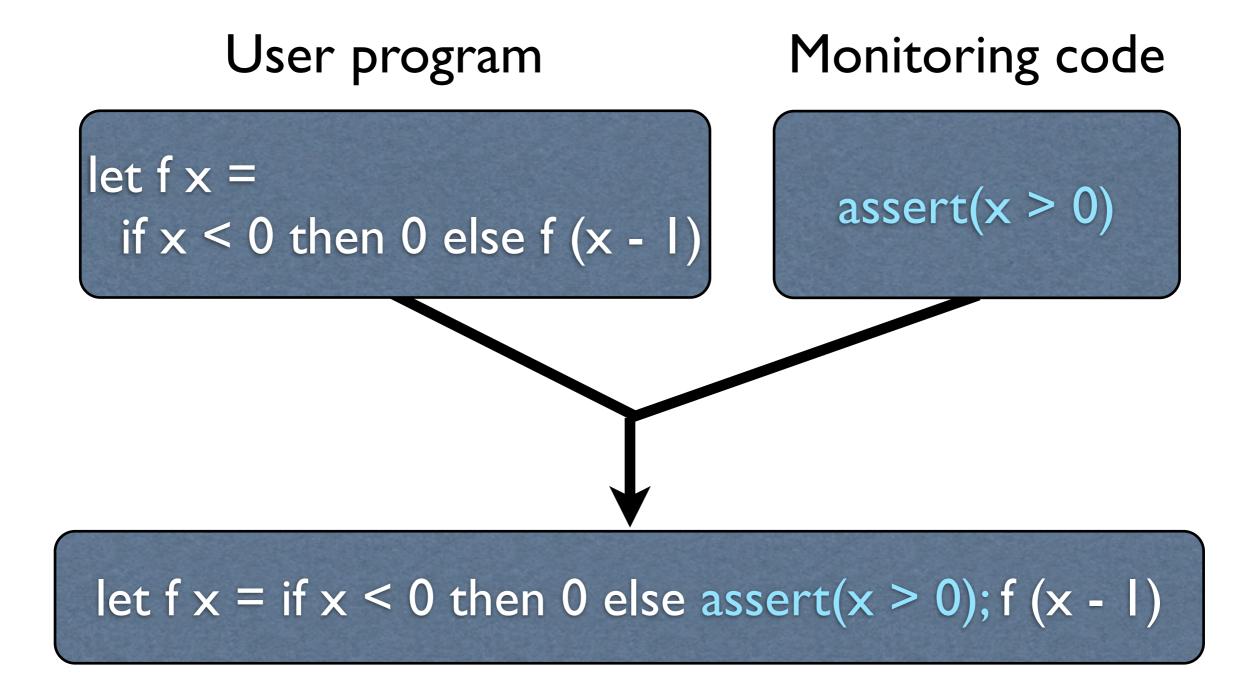
User program

Monitoring code

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



Product construction



Monitor specification

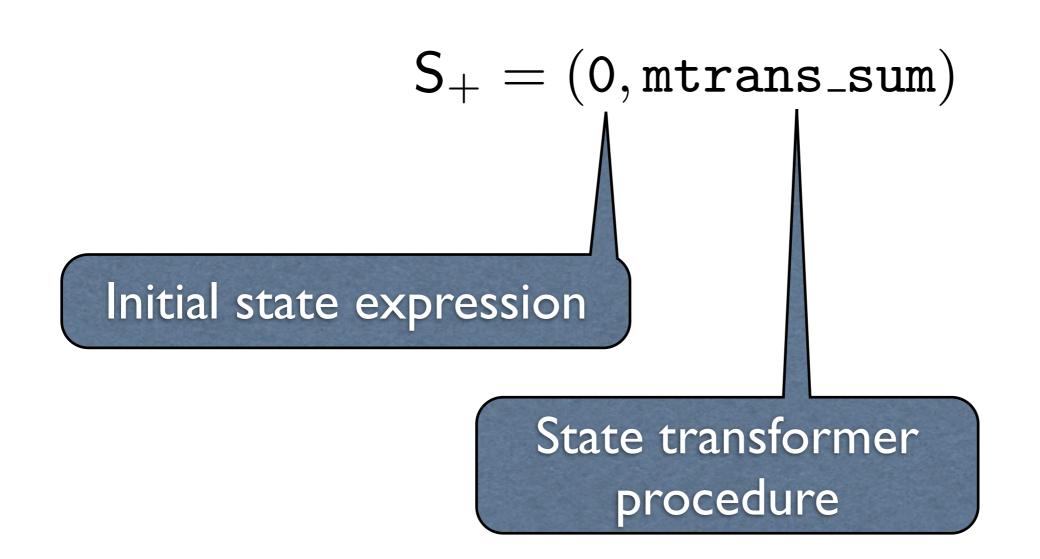
 $S_+ = (\texttt{0}, \texttt{mtrans_sum})$

Monitor specification

$$S_+ = (0, mtrans_sum)$$

Initial state expression



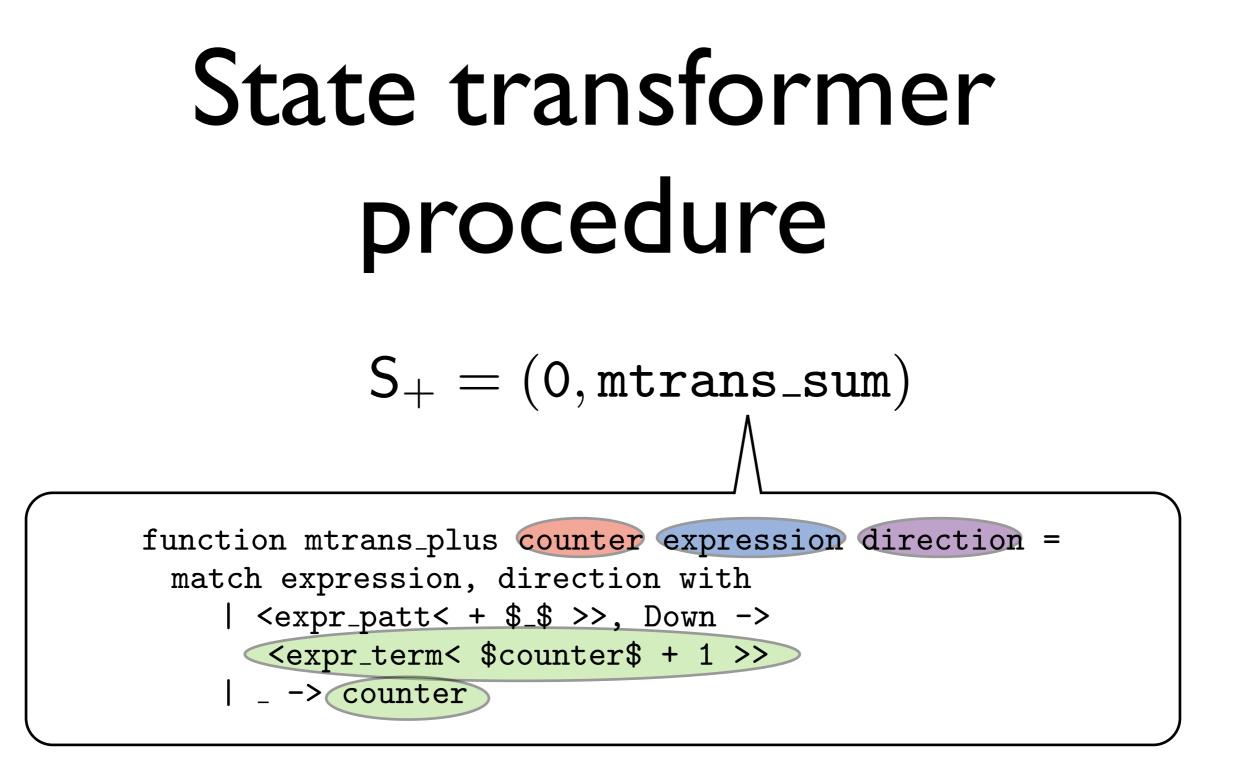


State transformer procedure $S_+ = (0, mtrans_sum)$ function mtrans_plus counter expression direction = match expression, direction with | <expr_patt< + \$_\$ >>, Down -> <expr_term< \$counter\$ + 1 >>

 $|_{-}$ -> counter

$$\label{eq:state_transformer} \begin{split} & \text{State transformer} \\ & \text{procedure} \end{split} \\ & \text{S}_+ = (0, \texttt{mtrans_sum}) \end{split}$$

(State X Expression X Direction) → Monitoring code



(State X Expression X Direction) → Monitoring code

Product of I+2 and M_+

1 + 2

Product of I+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 I+2 and M_+

let x_plus =

 $(fun x_{11} x_{12} s_{1_pre} \rightarrow x_{11} + x_{12}, s_{1_pre})$ 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
    l 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
    l 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
    l 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
    l 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	breadtha	Breadth first tree fold function.	LinIneq
13	app_tail 🛛	Apply function to tail of list.	LinIneq
14	compose_listo	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 \heartsuit	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 62	simple-rec	A simple recursive function.	Term
	sum	The sum of the first n naturals.	Term

>Higher order benchmark.
 ▲Benchmark on trees.

Benchmark on file descriptors.

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 firstorder functional programs. In *POPL*, 2003.
- [4] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, 2009.

1 2 3 4 5 6 7 8	rev_aux reverse ins	Description List reversal helper function. List reversal function. Ordered insert function	LinIneq LinIneq
3 4 5 6 7 8	ins		LinIneq
4 5 6 7 8		Ordered insert function	
5 6 7 8	-1	Ofdered hisert function	LinIneq
6 7 8	clone	Create a pair of copies of a list.	LinIneq
7 8	tpo	Insert an element in every third position of a list.	LinIneq
8	sec	Remove every third element from a list.	LinIneg
	tos	Replace each third element of a list.	LinIneg
	append	List append.	LinIneq
9	split_by	Split list.	LinIneq
10	append	List append.	LinIneg
11	breadth_aux.	Breadth first tree fold auxiliary function.	LinIneg
12	breadtha	Breadth first tree fold function.	LinIneg
13	app_tail~	Apply function to tail of list.	LinIneg
14	compose_list♡	Martin Hofmann's composing (or folding) all functions in a list	LinIneg
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_allo	Function for_all from list.ml.	LinIneq
22	rev_mapo	Function rev_map from list.ml.	LinIneq
23	itero	Function iter from list.ml.	LinIneq
24	fold_righto	Function fold_right from list.ml.	LinIneq
25	fold_leftv	Function fold_left from list.ml.	LinInea
26	mapo	Function map from list.ml.	LinIneq
20	mapo memq	Function memory from list.ml.	LinIneq
28	memq mem_assoc	Function mem_assoc from list.ml.	LinIneq
20 29	-		
30	mem_assq	Function mem_assq from list.ml.	LinIneq
31	remove_assoc	Function remove_assoc from list.ml.	LinIneq
	find_all♡	Function find_all from list.ml.	LinIneq
32	partition	Function partition from list.ml.	LinIneq
33 34	split	Function split from list.ml.	LinIneq
	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
50	rev_append	Append a list reversed.	Term
59	rev_merge	Merge two lists.	Term
59 60			Term
	simple-rec	A simple recursive function.	Term
60	simple-rec sum	A simple recursive function. The sum of the first n naturals.	Term

Benchmark on trees.
 Benchmark on file descriptor.

Evaluation: benchmarks

Recursion

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

ſ	#	Name	Description	Class
ł	1	rev_aux	List reversal helper function.	LinInea
	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.	LinIneg
	6	sec	Remove every third element from a list.	LinIneg
	7	tos	Replace each third element of a list.	LinIneq
1	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	breadths	Breadth first tree fold function.	LinIneq
	13	app_tail♡	Apply function to tail of list.	LinIneq
	14	compose_list \heartsuit	Martin Hofmann's composing (or folding) all functions in a list	LinIneq
	15 16	processfile	Open, read, and close one file.	Set Set
	10	processfile	Open, read, and close one file. Manipulate two files at the same time.	Set
	18	processtwofiles.	Manipulate two files at the same time, version 2.	Set
	10	mem	Function mem from list.ml.	LinIneg
	20	exists♡	Function exists from list.ml.	LinIneq
	20	for_allo	Function for_all from list.ml.	LinIneq
	22	rev_map♡	Function rev_map from list.ml.	LinIneq
	23	iterv	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.	LinIneg
	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 33	partition v	Function partition from list.ml.	LinIneq
	33 34	split remove_assq	Function split from list.ml. Function remove_assq from list.ml.	LinIneq 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.	LinInea
	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 47	list_of_leaves.	Left to right list of leaves of a tree.	LinIneq
	47	left_path a twice	Given a tree, the path from the leftmost leave to the root. Duplicate every element of a list.	LinIneq, LinIneq
	48 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_treev.	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 60	rev_append	Append a list reversed. Merge two lists.	Term Term
	61	rev_merge simple-rec	A simple recursive function.	Term
	62	sum	The sum of the first n naturals.	Term
l	02	- Sum	The sum of the mat <i>n</i> naturals.	1 10111

◦Higher order benchmark. ♣Benchmark on trees.

Benchmark on file descriptors

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		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		List append.	LinIneq
9		Split list.	LinIneq
10		List append.	LinIneq
11		Breadth first tree fold auxiliary function.	LinIneq
12		Breadth first tree fold function.	LinIneq
13	111 - 11	Apply function to tail of list.	LinIneq
14		Martin Hofmann's composing (or folding) all functions in a list	LinIneq
15		Open, read, and close one file.	Set
16		Open, read, and close one file.	Set
17		Manipulate two files at the same time.	Set
18	1	Manipulate two files at the same time, version 2.	Set
19		Function mem from list.ml.	LinIneq
20		Function exists from list.ml.	LinIneq
21		Function for_all from list.ml.	LinIneq
22		Function rev_map from list.ml.	LinIneq
23		Function iter from list.ml.	LinIneq
24	= 0	Function fold_right from list.ml.	LinIneq
25		Function fold_left from list.ml.	LinIneq
26	t.	Function map from list.ml.	LinIneq
27		Function memq from list.ml.	LinIneq
28		Function mem_assoc from list.ml.	LinIneq
29		Function mem_assq from list.ml.	LinIneq
30		Function remove_assoc from list.ml. Function find_all from list.ml.	Linlneq
32			LinIneq
32		Function partition from list.ml.	LinIneq LinIneq
34		Function split from list.ml. Function remove_assq from list.ml.	LinIneq
35		Auxiliary function for depth first tree fold to list.	LinIneq
36		Function for depth first tree fold to list.	LinIneq
37		Function for depth first tree fold to list, version 2.	LinIneq
38		Function for constructing a tree from a list.	LinIneq
39		Function for folding a tree into a list, version 2.	LinIneq
40		Function for folding a tree into a list.	LinIneq
41		Quicksort.	LinIneq
42		Insertion sort.	LinIneq
43		Open, read, and close one file, version 2.	Set
44	1	Open, read, and close one file, version 2.	Set
4		Martin Hofmann's composing (or folding) all functions in a list	LinIneq
46		Left to right list of leaves of a tree.	LinIneq
47		Given a tree, the path from the leftmost leave to the root.	LinIneq,
48		Duplicate every element of a list.	LinIneq
49		Add a number if it is a positive integer.	LinIneq
50		Add a number if it is a positive integer.	LinIneq
51		Pre-order list fold.	LinIneq
52	2 write_byte.	Write byte to file.	Set
53		The Ackermann function.	Term
54		Chop the first n elements of a list.	Term
55		An algebraic data type recursive manipulation.	Term
56		Fold a pair of lists.	Term
57		The McCarthy 91 function.	Term
58		Recursive definition of multiplication.	Term
59		Append a list reversed.	Term
		Manage true lists	Term
60	0	Merge two lists.	
60 61 62	l simple-rec	A simple recursive function. The sum of the first <i>n</i> naturals.	Term Term

Higher order benchmark.
 Benchmark on trees.
 Benchmark on file descriptors.

Demo

Conclusion

- Monitoring
- Product construction
- Evaluation