LEAP
A Tool for the Parametrized Verification of Concurrent Datatypes

Alejandro Sánchez[1] César Sánchez[1,2]

[1]IMDEA Software Institute, Spain

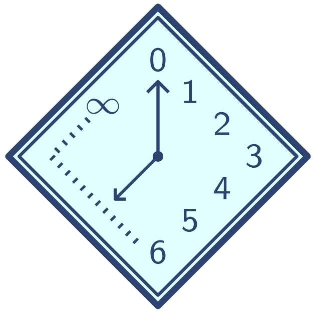
[2]Institute for Information Security (CSIC), Spain

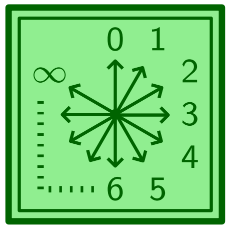
CAV'14, Vienna, 21 July 2014

LEAP: Objectives

LEAP: Objectives

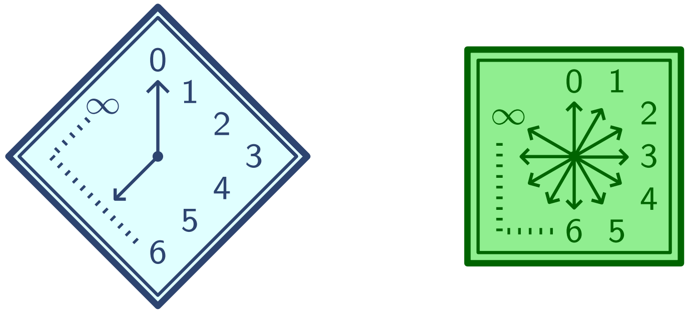
Temporal Properties



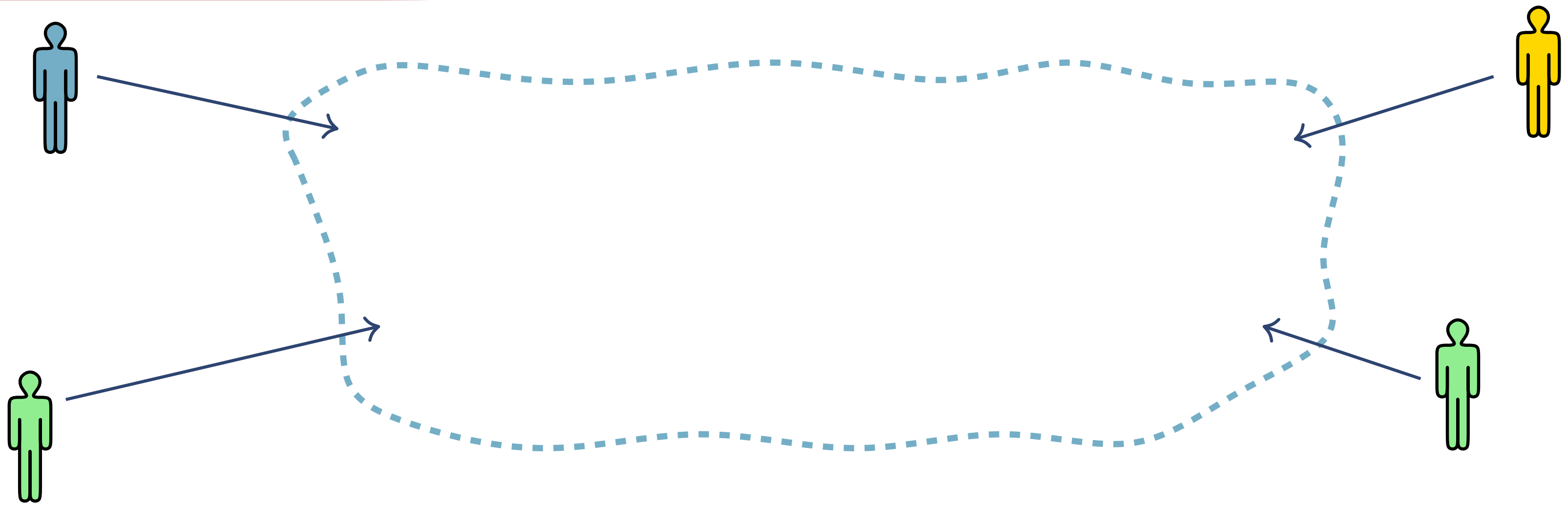


LEAP: Objectives

Temporal Properties

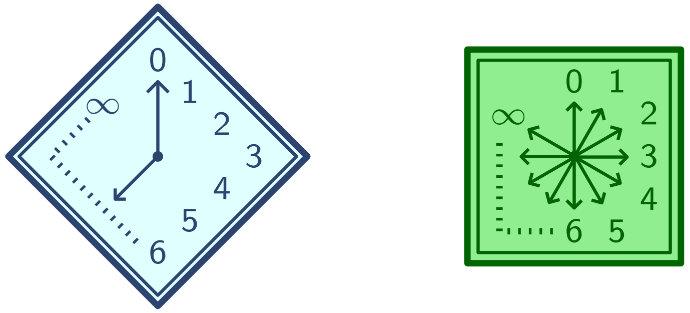


Concurrent Datatypes

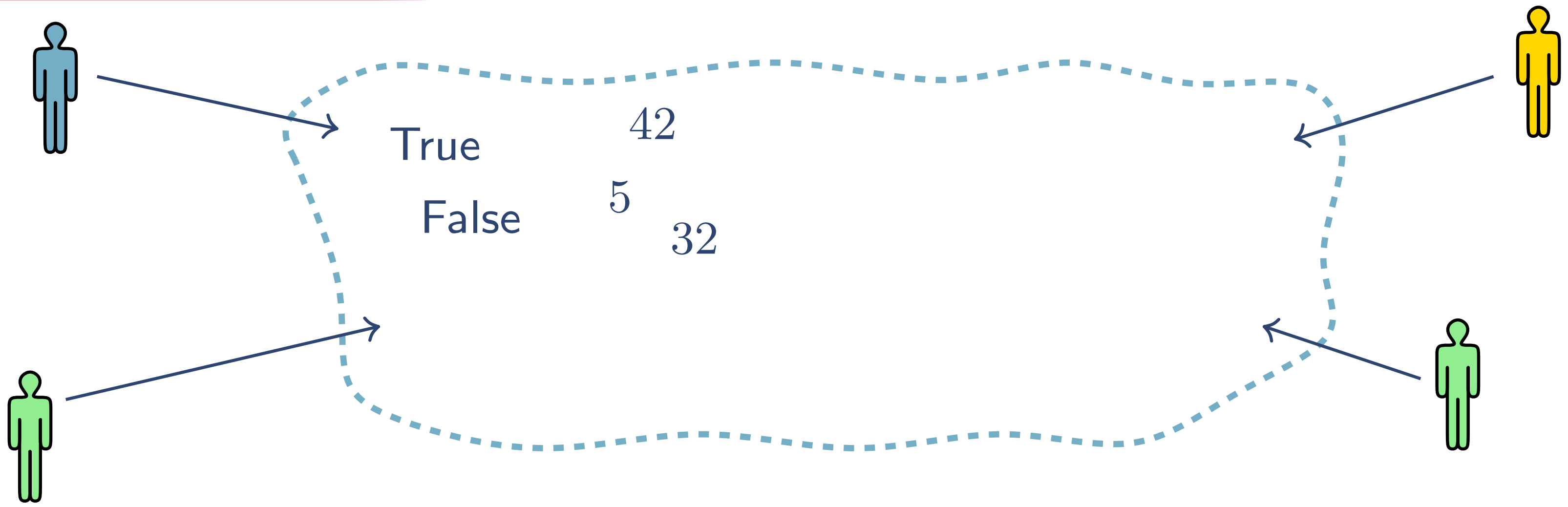


LEAP: Objectives

Temporal Properties

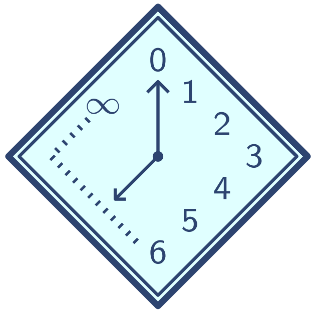


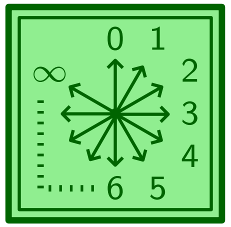
Concurrent Datatypes



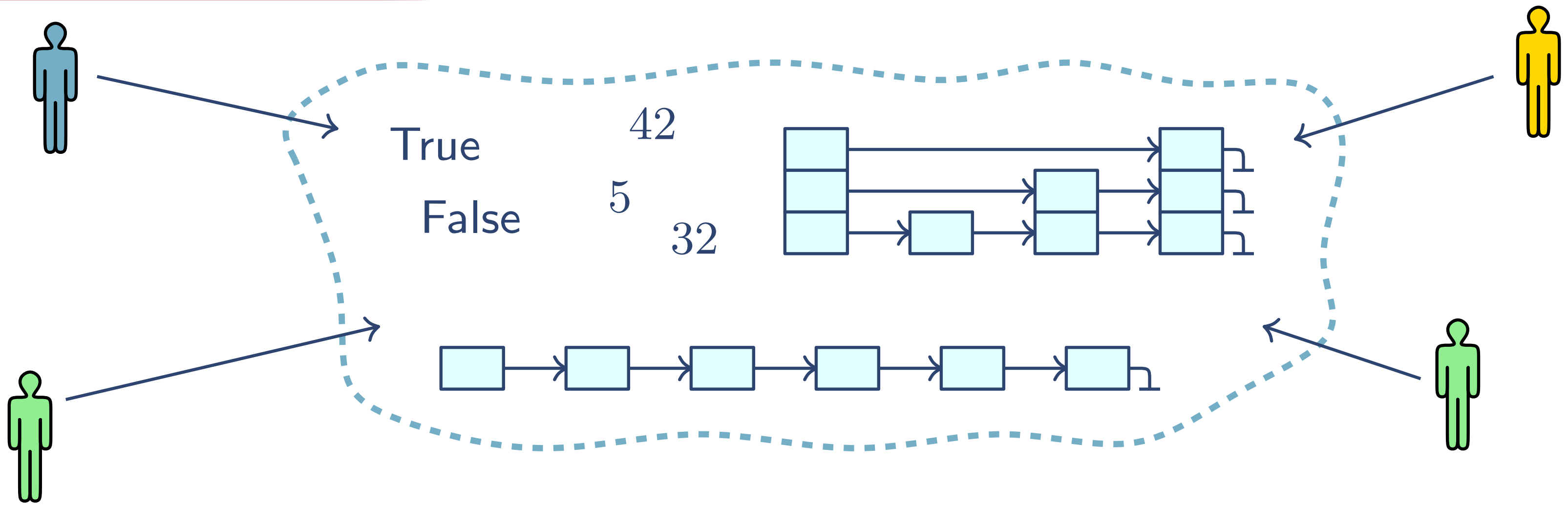
LEAP: Objectives

Temporal Properties



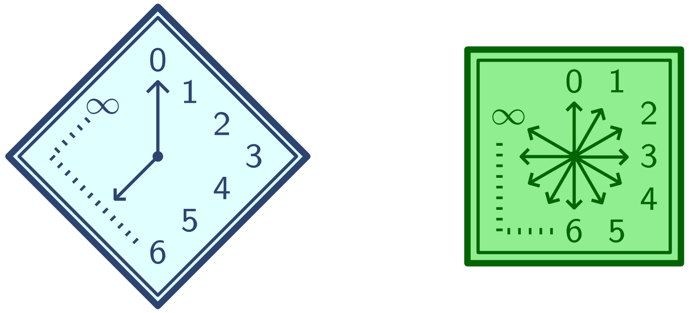


Concurrent Datatypes

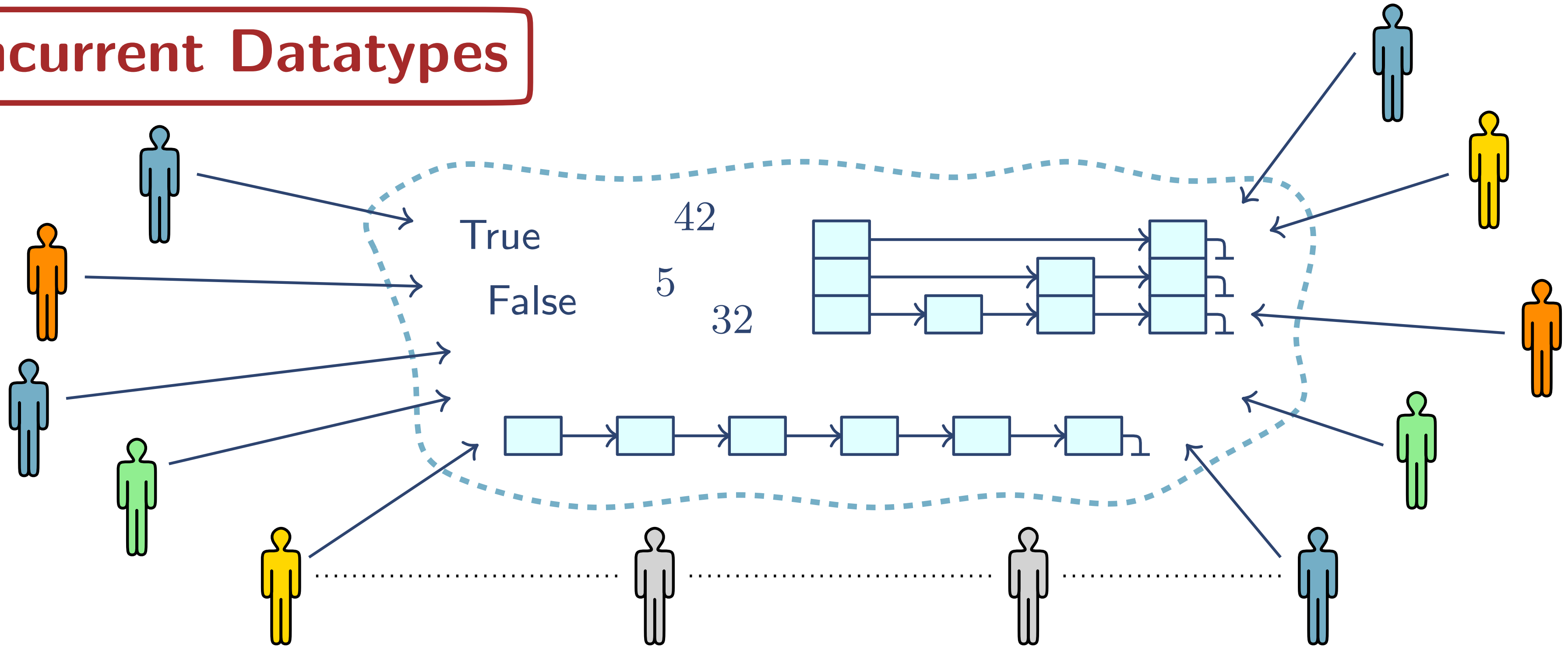


LEAP: Objectives

Temporal Properties



Concurrent Datatypes



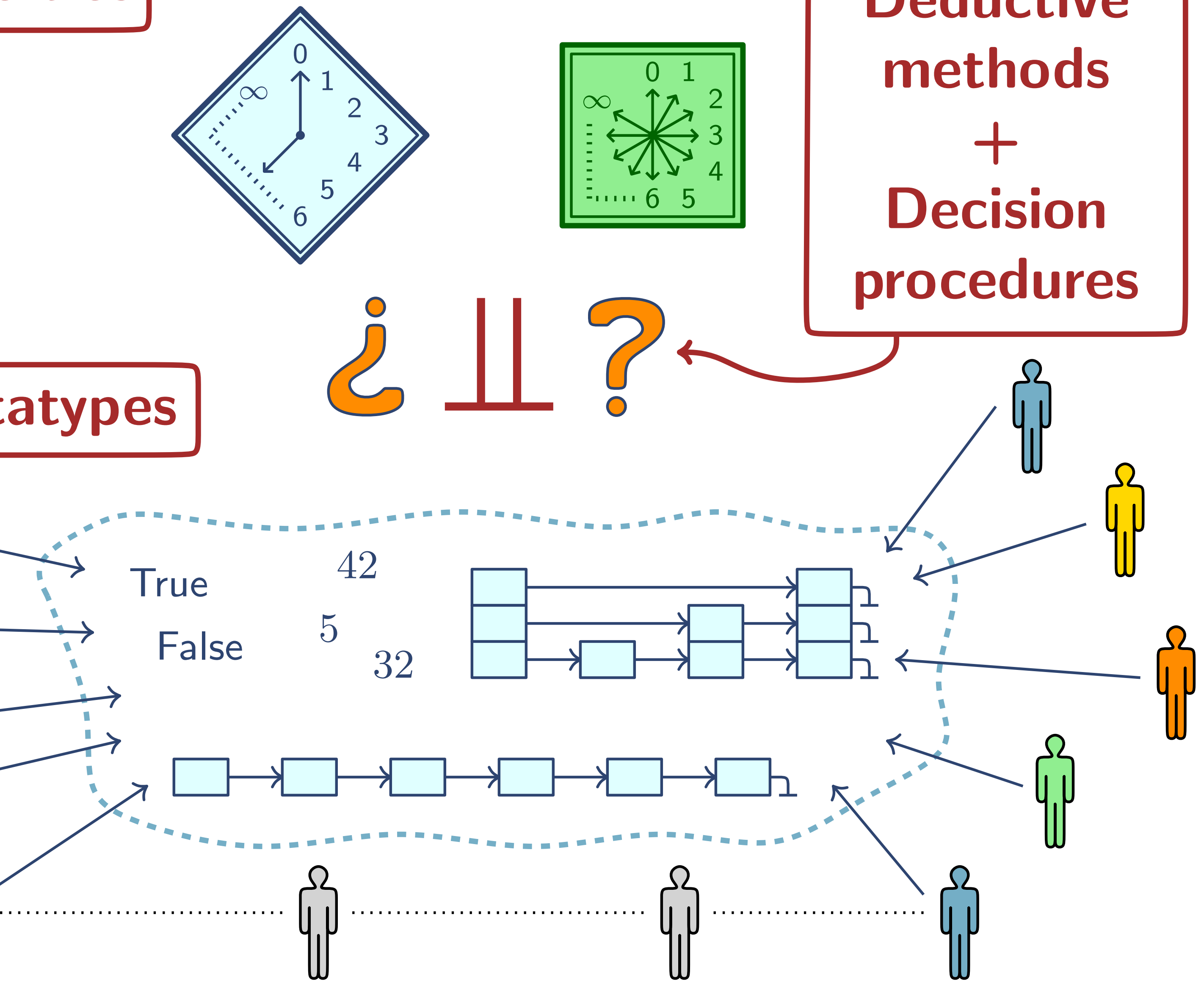
Parametrized Verification

LEAP: Objectives

Temporal Properties

Deductive methods + Decision procedures

Concurrent Datatypes



Parametrized Verification

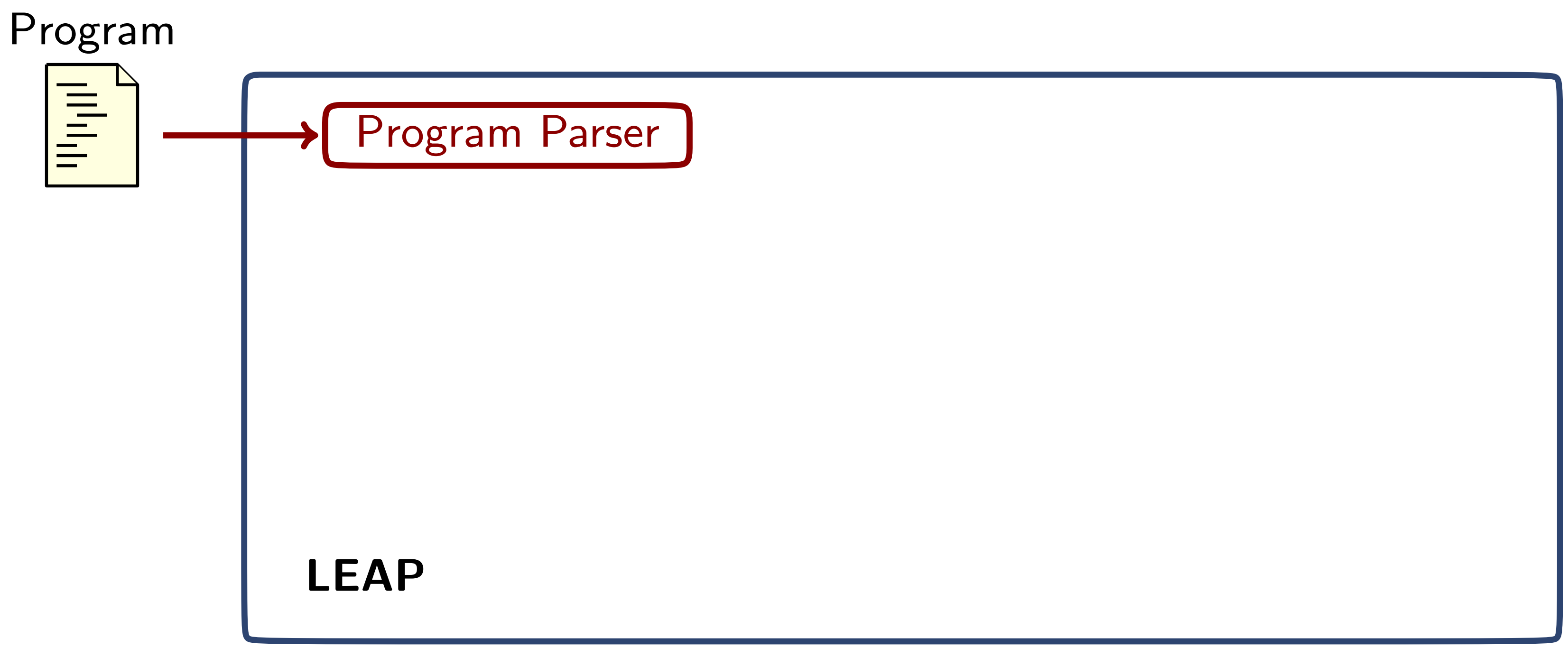
LEAP: Description

- **Deductive** theorem prover
- Aims verification of **temporal properties**
- Designed to verify **concurrent datatypes**
- Specific for **parametrized systems**
- Automation based on **decision procedures**
- Built on top of state-of-the-art **SMT solvers**
- Targets both **safety** and **liveness** verification

LEAP: Structure

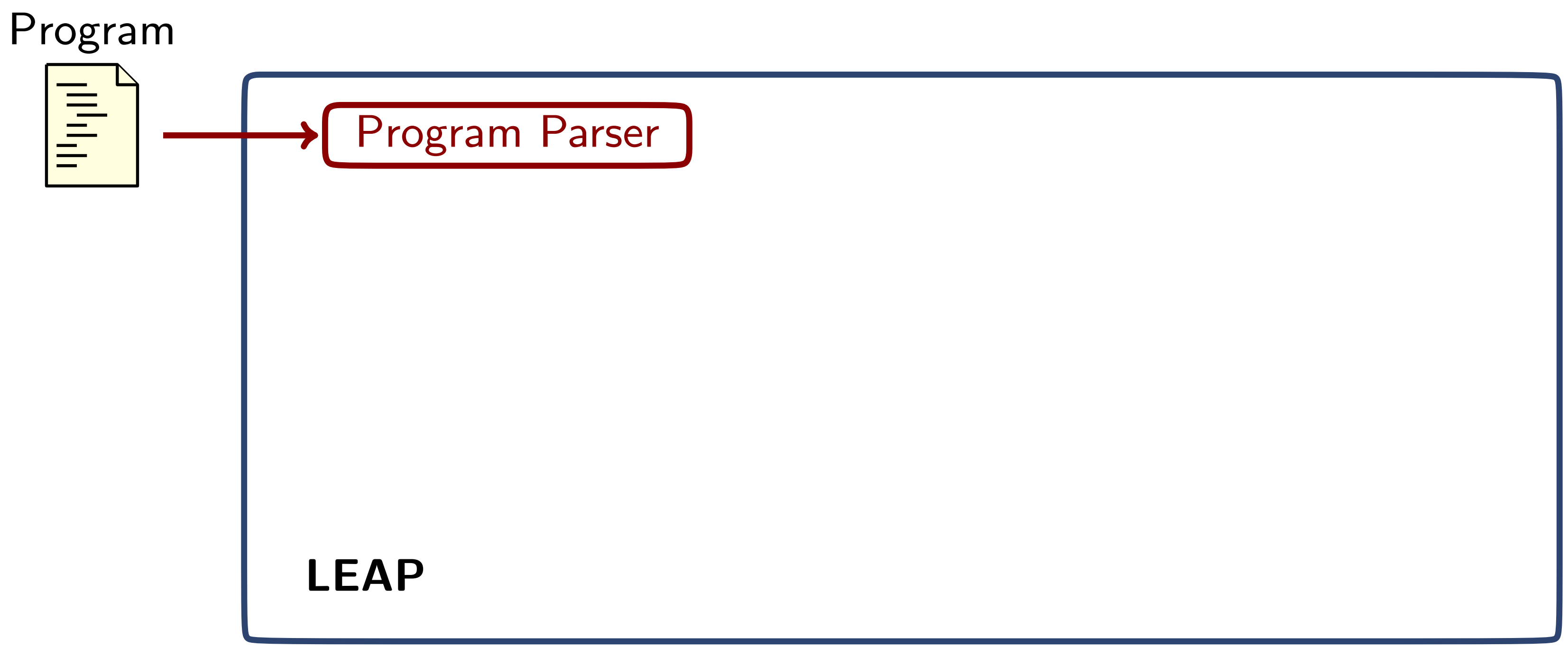
LEAP

LEAP: Structure



```
    global
      addr head, tail; ghost addrSet region;
    procedure insert (e:elem)
      addr prev, curr, aux;
    begin
 1: prev := head;
 2: prev->lock;
 3: curr := prev->next;
 4: curr->lock;
 5: while curr->data < e do
          ......
          ......
11: end while
12: if curr != null /\curr->data > e then
13:     aux := malloc(e,null,#);
14:     aux->next := curr;
    :connect
15:     prev->next := aux
          $region := region Union {aux};$
16: end if
    ......
20: return(inserted)
    end procedure
```

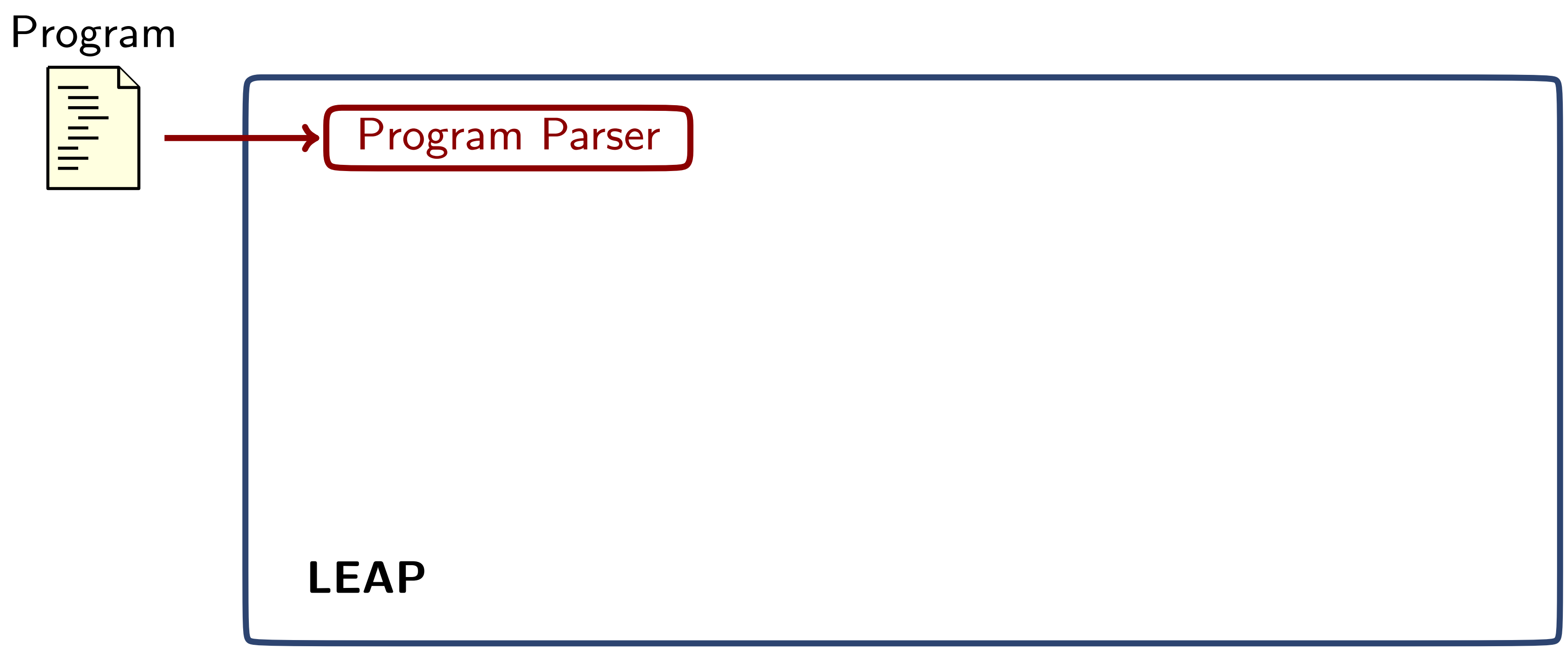
LEAP: Structure



```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
    ......
    ......
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

Conditionals, loops

LEAP: Structure

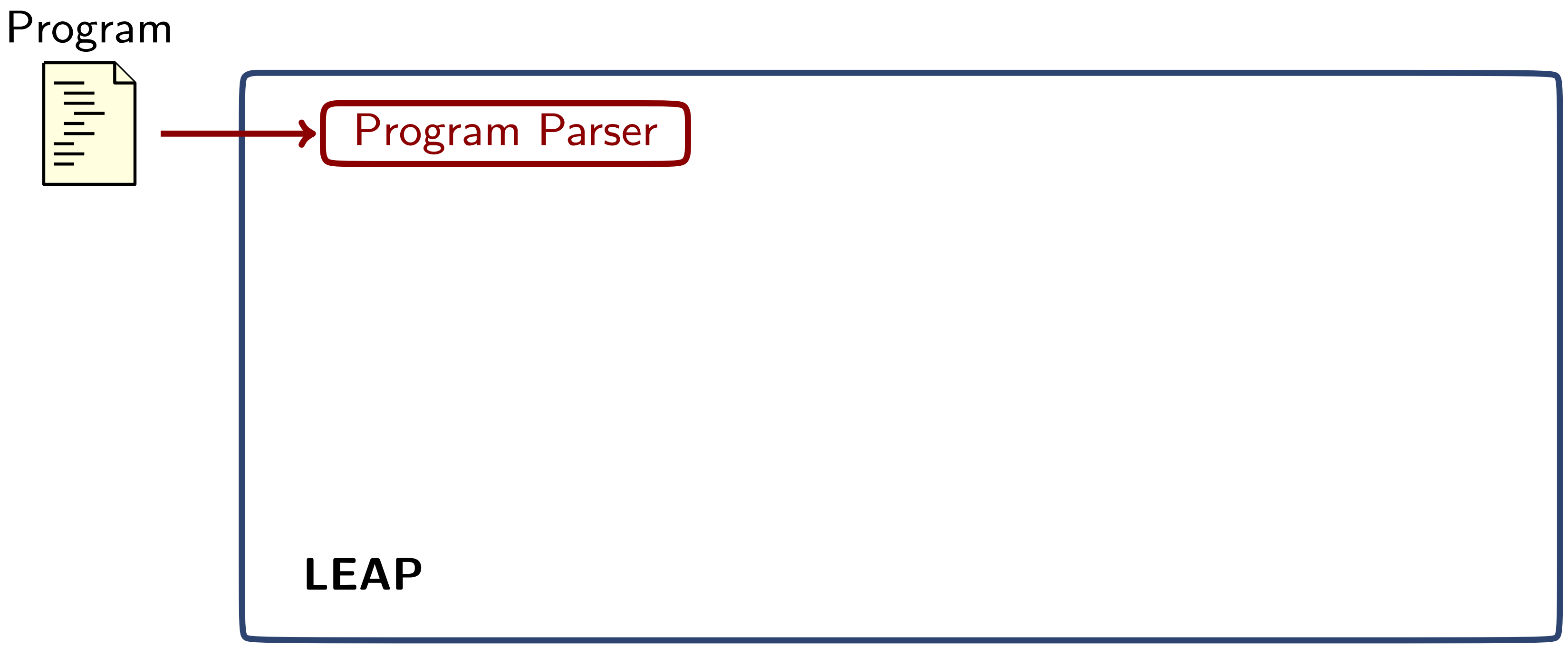


```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
      ......
      ......
end while
if curr != null /\curr->data > e then
    aux := malloc(e,null,#);
    aux->next := curr;
:connect
    prev->next := aux
     $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

Conditionals, loops

Pointers

LEAP: Structure



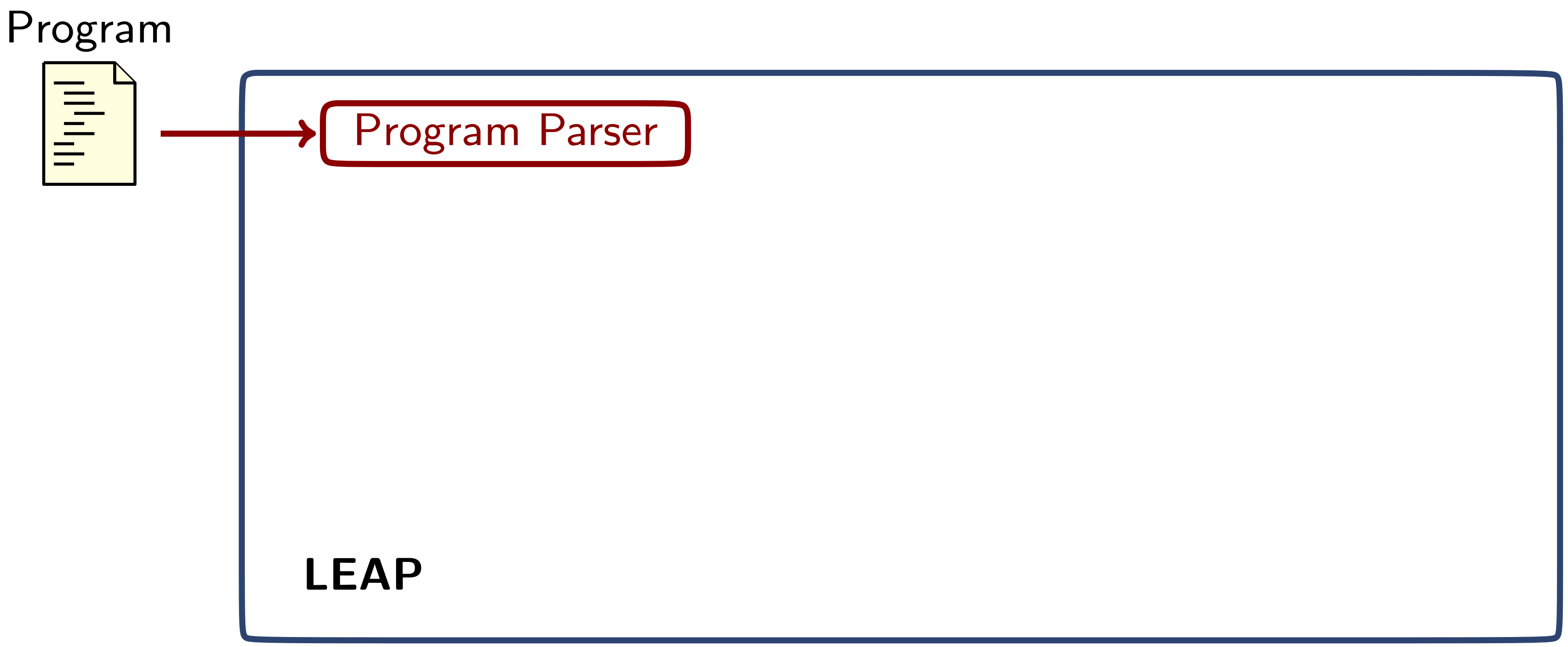
```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
      ......
      ......
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

Conditionals, loops

Pointers

Function calls (no recursion)

LEAP: Structure



```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
    ......
    ......
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

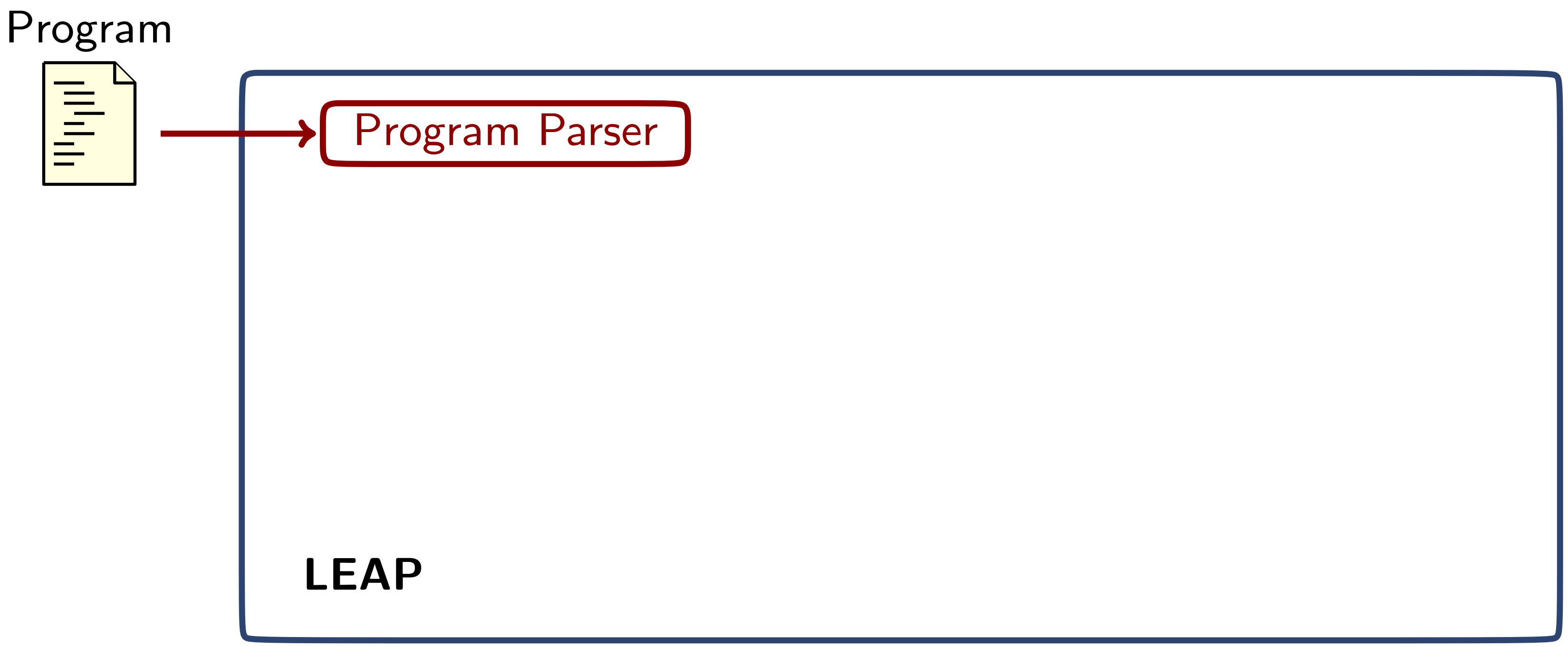
Conditionals, loops

Pointers

Function calls (no recursion)

Memory allocation

LEAP: Structure



```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
    ......
    ......
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

Conditionals, loops

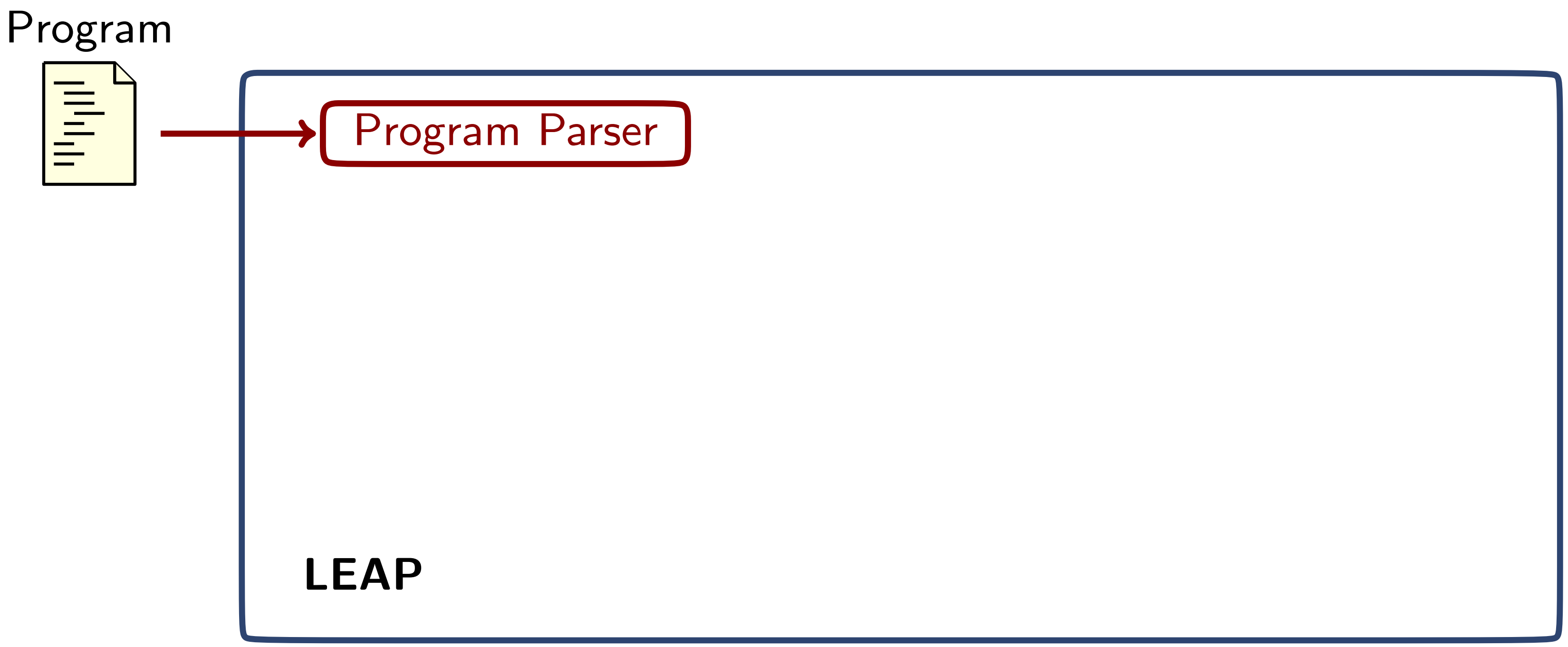
Pointers

Function calls (no recursion)

Memory allocation

Atomic sections

LEAP: Structure



```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
    ......
    ......
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
......
return(inserted)
end procedure
```

Conditionals, loops

Pointers

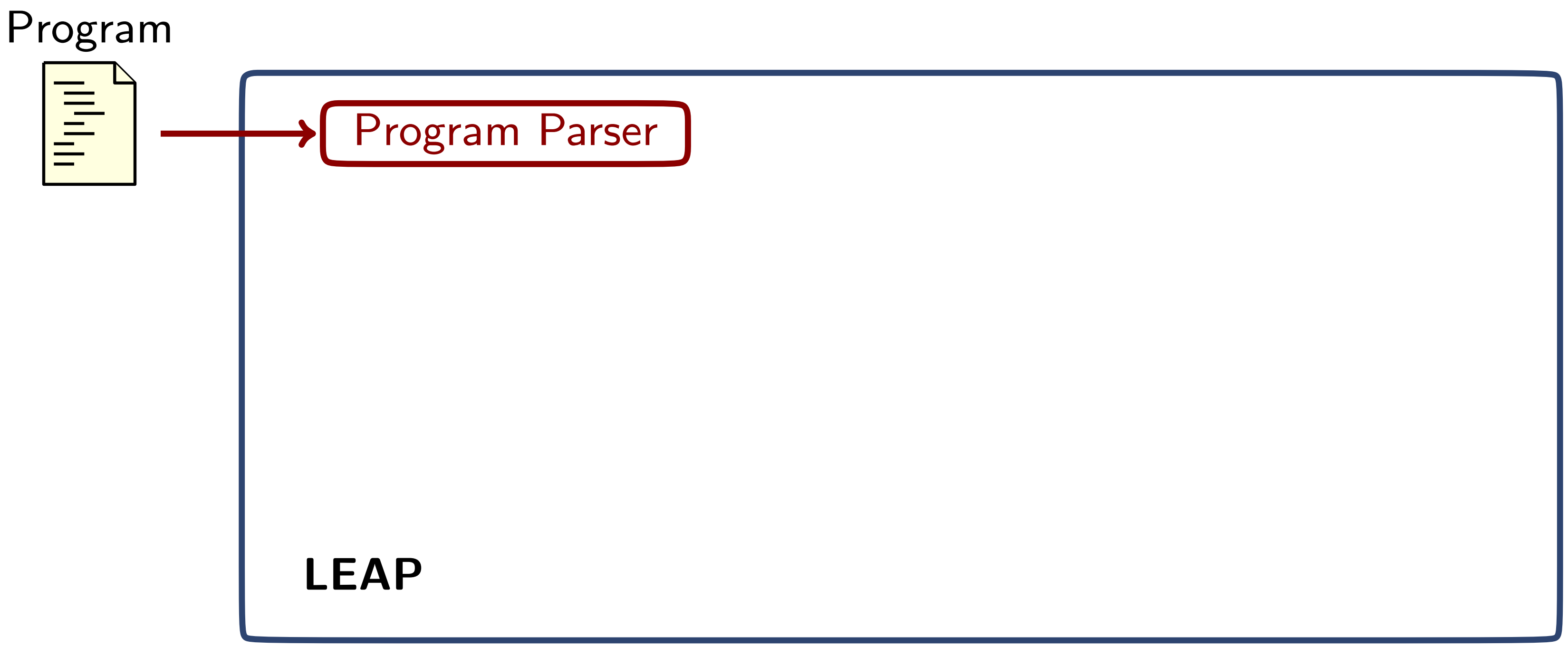
Function calls (no recursion)

Memory allocation

Atomic sections

Ghost code

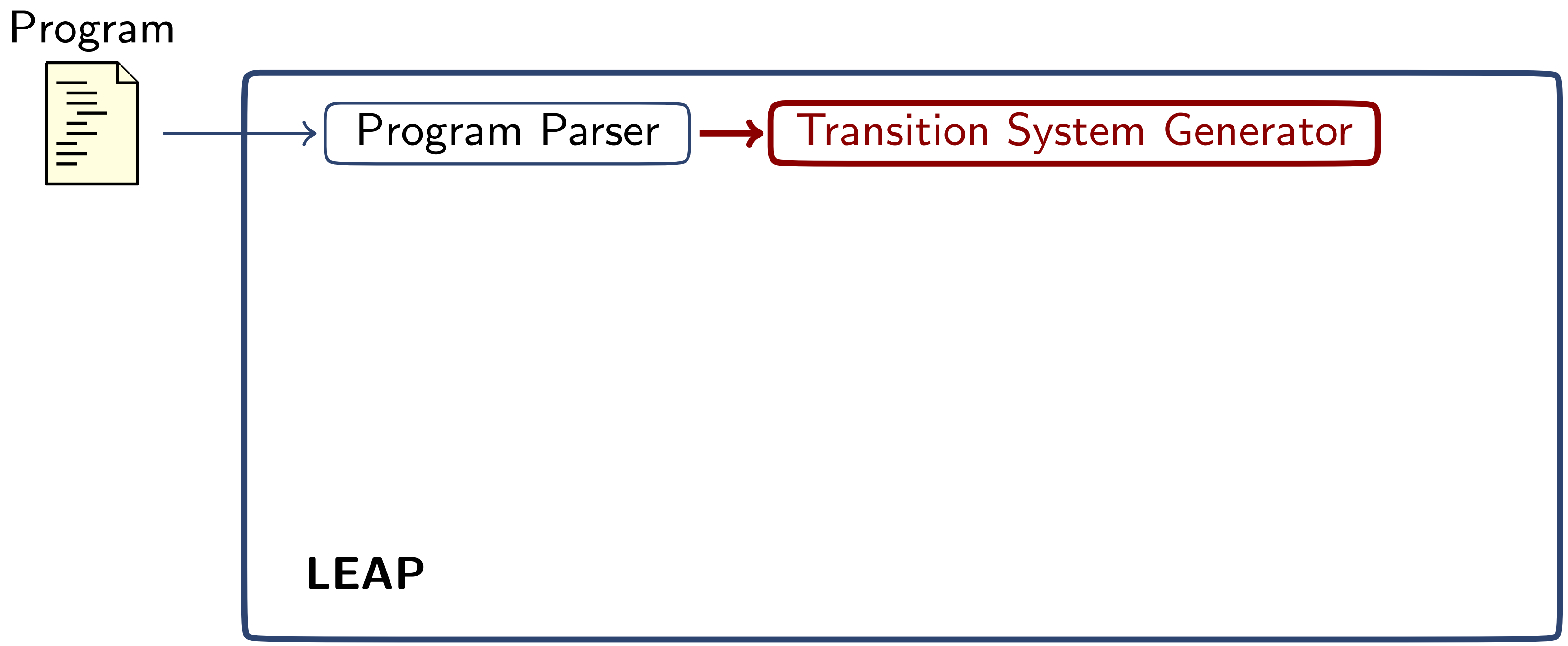
LEAP: Structure



```
global
  addr head, tail; ghost addrSet region;
procedure insert (e:elem)
  addr prev, curr, aux;
begin
prev := head;
prev->lock;
curr := prev->next;
curr->lock;
while curr->data < e do
      ⋮
end while
if curr != null /\curr->data > e then
   aux := malloc(e,null,#);
   aux->next := curr;
:connect
   prev->next := aux
    $region := region Union {aux};$
end if
 ⋮
return(inserted)
end procedure
```

- Conditionals, loops
- Pointers
- Function calls (no recursion)
- Memory allocation
- Atomic sections
- Ghost code
- Location labeling

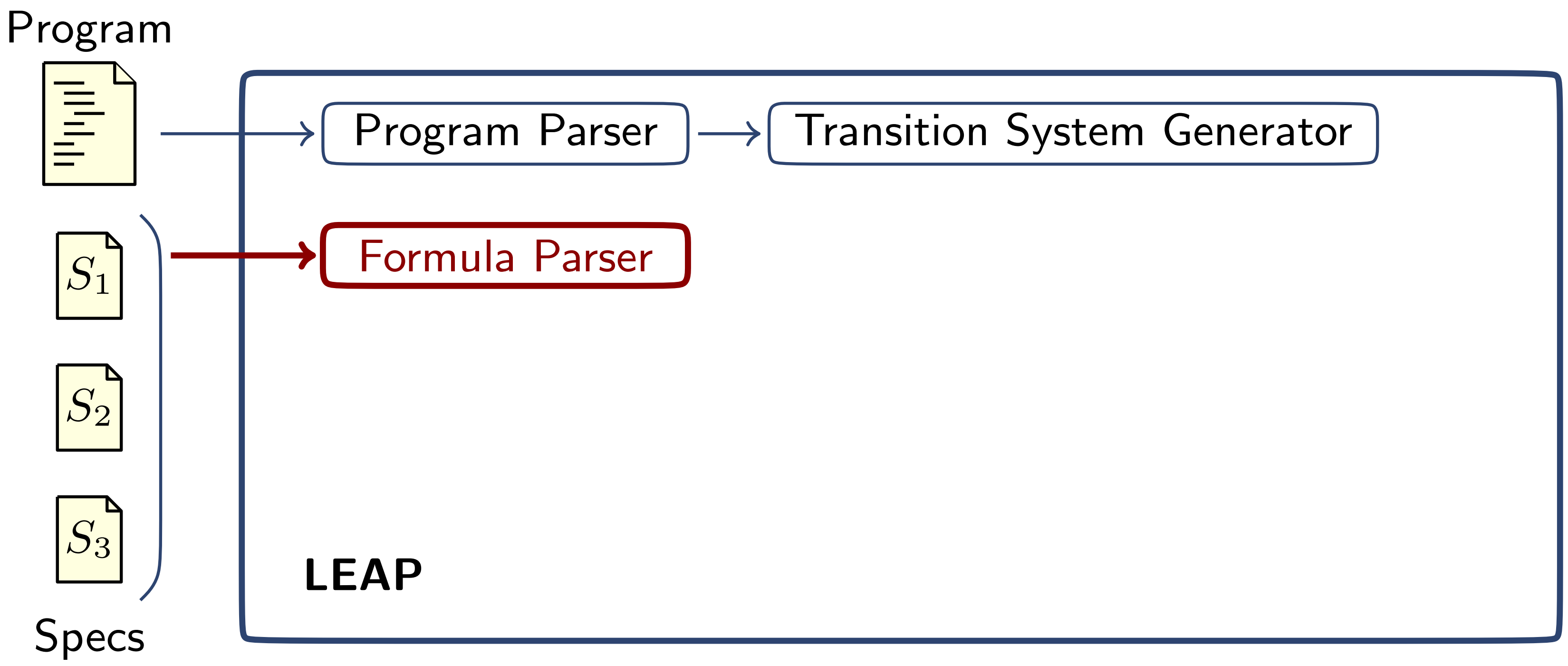
LEAP: Structure



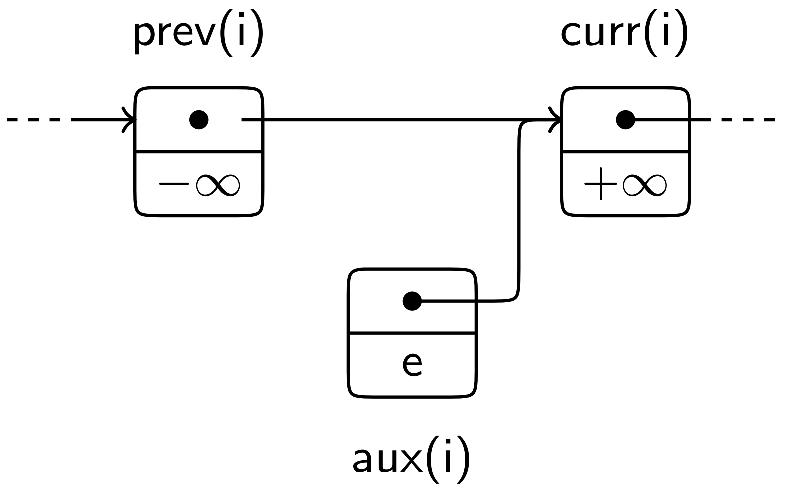
```
    global
      addr head, tail; ghost addrSet region;
    procedure insert (e:elem)
      addr prev, curr, aux;
    begin
 1: prev := head;
 2: prev->lock;
 3: curr := prev->next;
 4: curr->lock;
 5: while curr->data < e do
        ......
        ......
11: end while
12: if curr != null /\curr->data > e then
13:    aux := malloc(e,null,#);
14:    aux->next := curr;
    :connect
15:    prev->next := aux
        $region := region Union {aux};$
16: end if
    ......
20: return(inserted)
    end procedure
```

$$S[N] = T_1 \parallel T_2 \parallel \cdots \parallel T_N$$

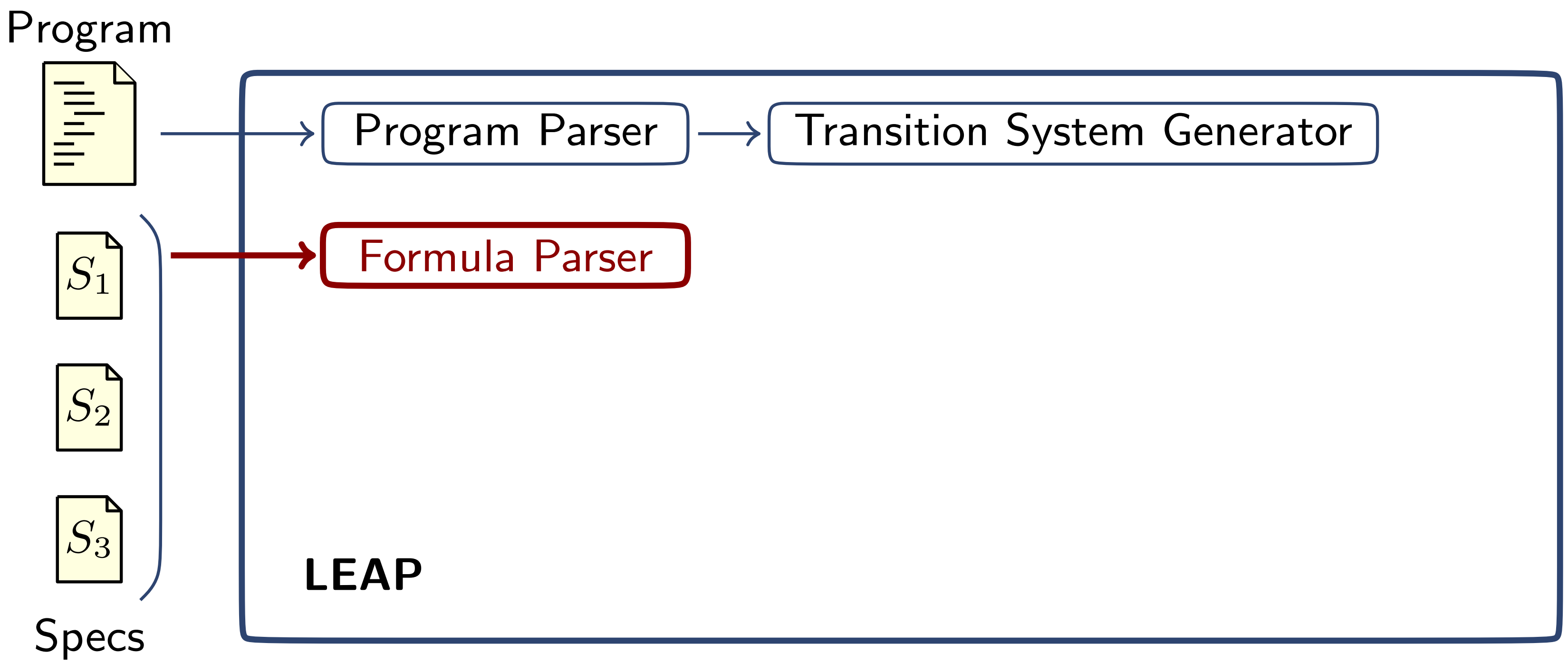
LEAP: Structure



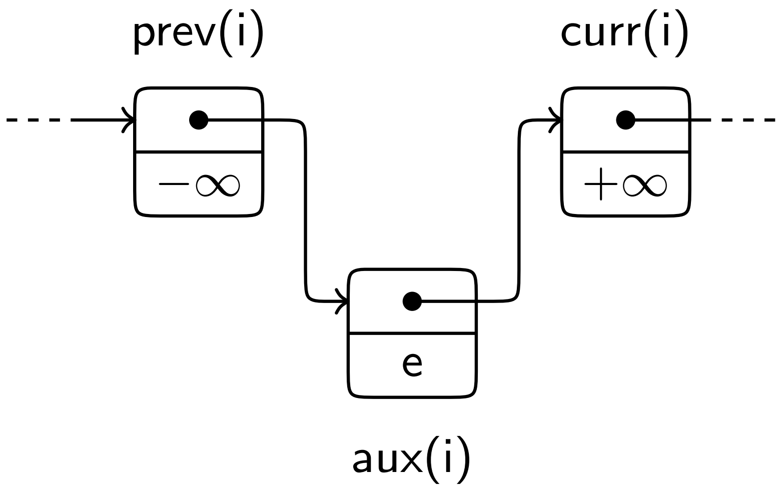
```
vars: tid i
specification [aux_ready] :
  @connect(i). ->
    (rd(heap, prev(i)).data < e        /\
     rd(heap, curr(i)).data > e        /\
     rd(heap, aux(i) ).data = e        /\
     rd(heap, prev(i)).next = curr(i)  /\
     rd(heap, aux(i) ).next = curr(i))
```



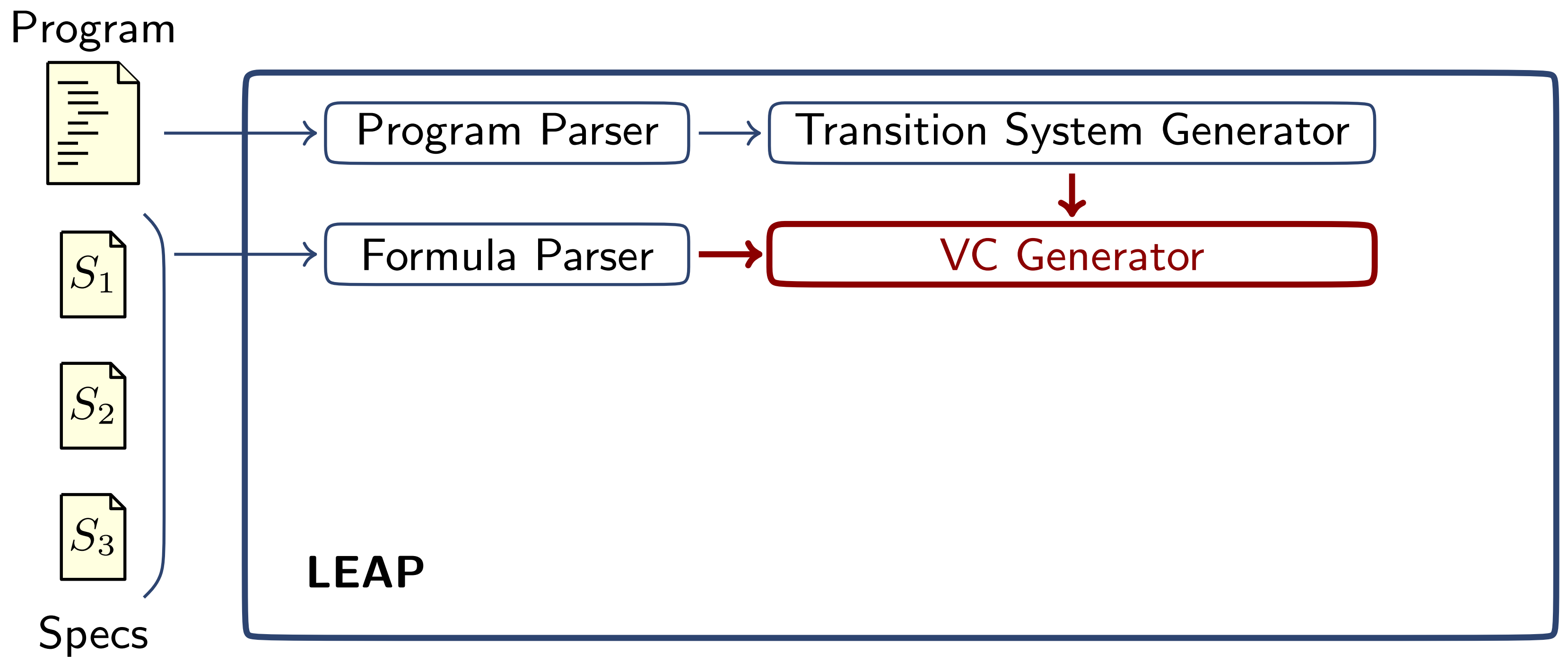
LEAP: Structure



```
vars: tid i
specification [aux_ready] :
  @connect(i). ->
    (rd(heap, prev(i)).data < e        /\
     rd(heap, curr(i)).data > e        /\
     rd(heap, aux(i) ).data = e        /\
     rd(heap, prev(i)).next = curr(i)  /\
     rd(heap, aux(i) ).next = curr(i))
```



LEAP: Structure

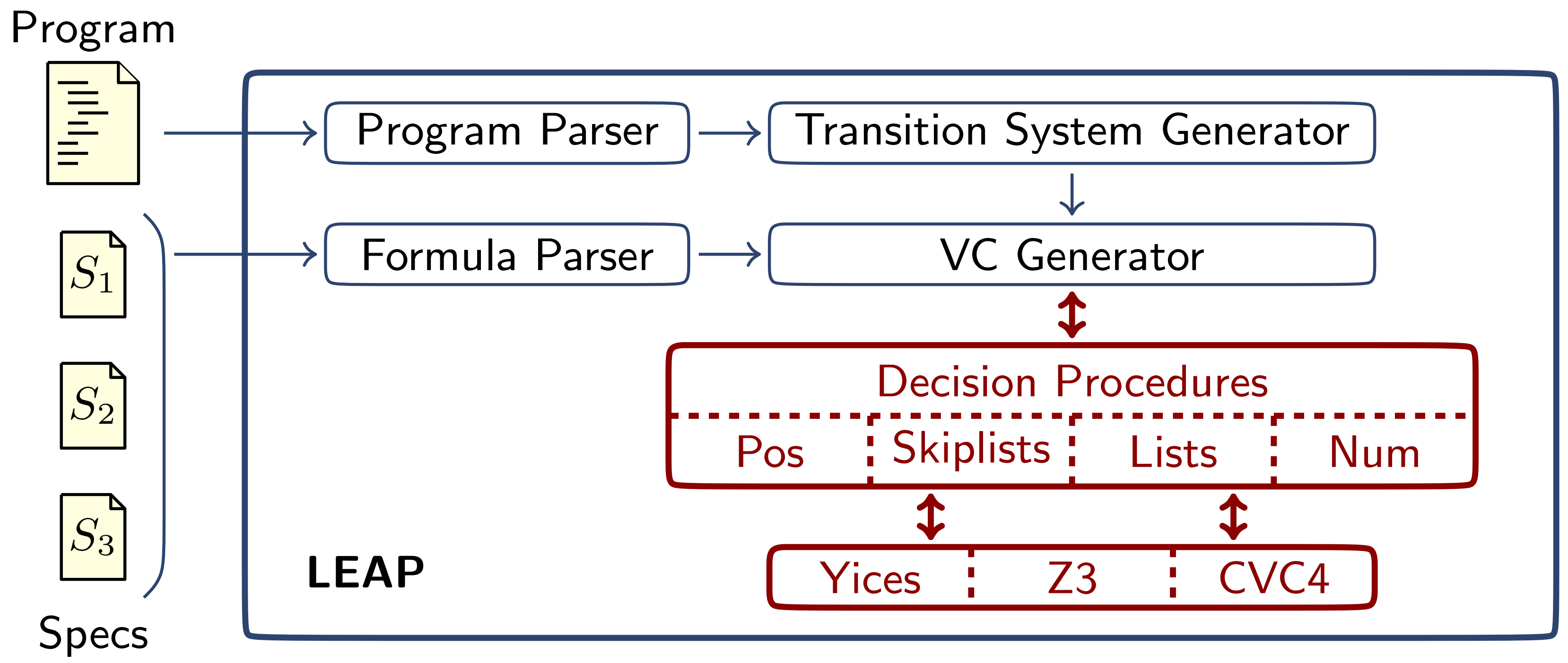


Each **VC** represents a **small-step** in the execution

All VC are **QF** as long as specs are QF

LEAP supports **heuristics and tactics** to aid verification

LEAP: Structure



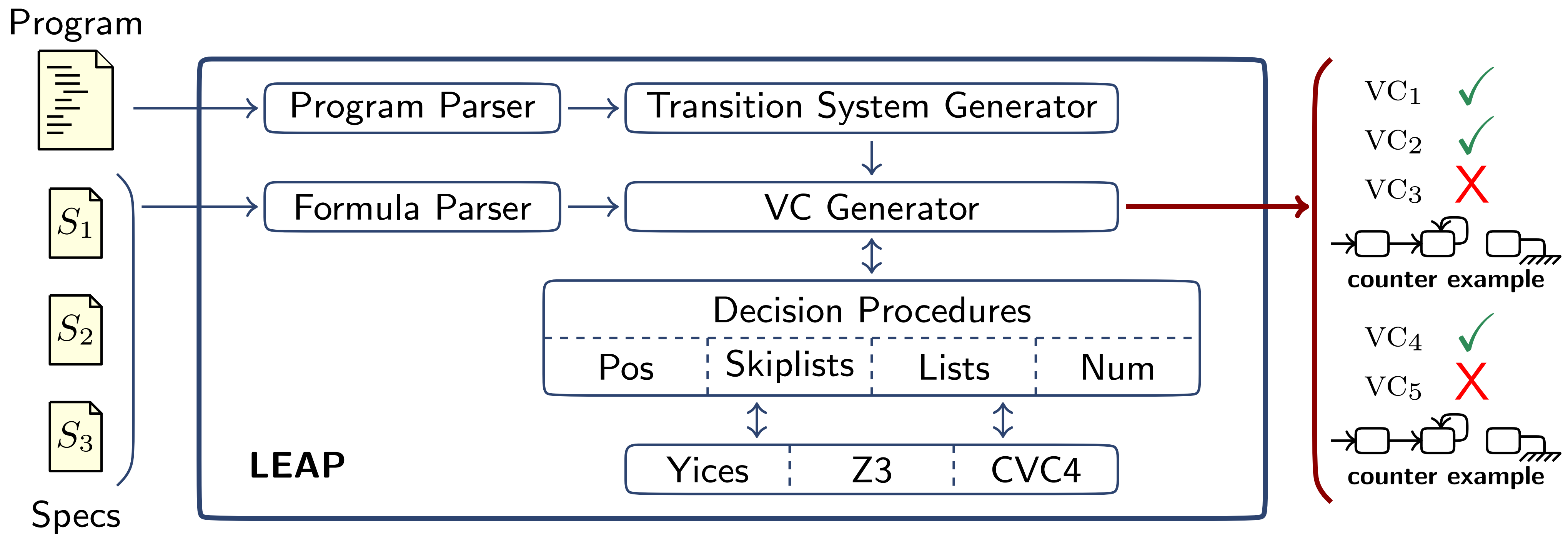
Each **VC** represents a **small-step** in the execution

All VC are **QF** as long as specs are QF

LEAP supports **heuristics and tactics** to aid verification

Check validity using **specialized decision procedures**

LEAP: Structure



All VCs are checked **valid**?

✓ ✗

Specification is verified

Generate small counter examples to aid the programmer

LEAP: Current Status

- **Full** implementation for **safety** properties
- **Ongoing** implementation for **liveness** properties [TIME'14]
- **Specialized decision procedures** for:
  - Presburger arithmetic with sets
  - Concurrent fine-grained and lock-free lists [ICFEM'10]
  - Concurrent fine-grained bounded height skiplists [NFM'11]
  - Unbounded skiplists [ATVA'14]
- We have **verified** structural and functional specifications of:
  - Mutual exclusion protocols based on integers and sets
  - Concurrent lock-coupling single-linked lists
  - Lock-free queues and stacks
  - Bounded and unbounded skiplists

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0				∞	1221.97	776.45	15.27	0.45
14	region$_3$	0				∞	27.50	17.36	0.34	0.58
15	next$_3$	0				∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

Heuristics are fundamental

LEAP: Experimental Results

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLinear	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	[illegible]	[illegible]	0.04	6.75
21	mutex	2	28	26	2	0.32	[illegible]	[illegible]	[illegible]	0.01
22	minticket	1	19	18	1	0.04	[illegible]	[illegible]	[illegible]	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

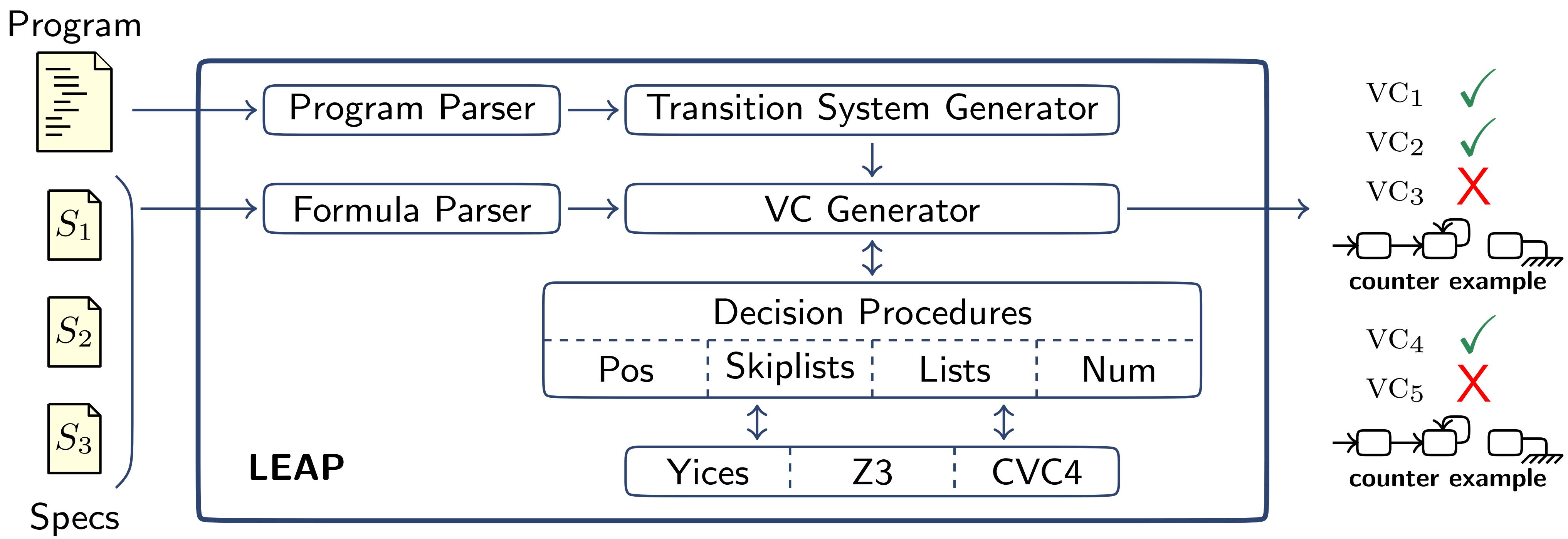
LEAP analysis time remains insignificant

LEAP: Experimental Results

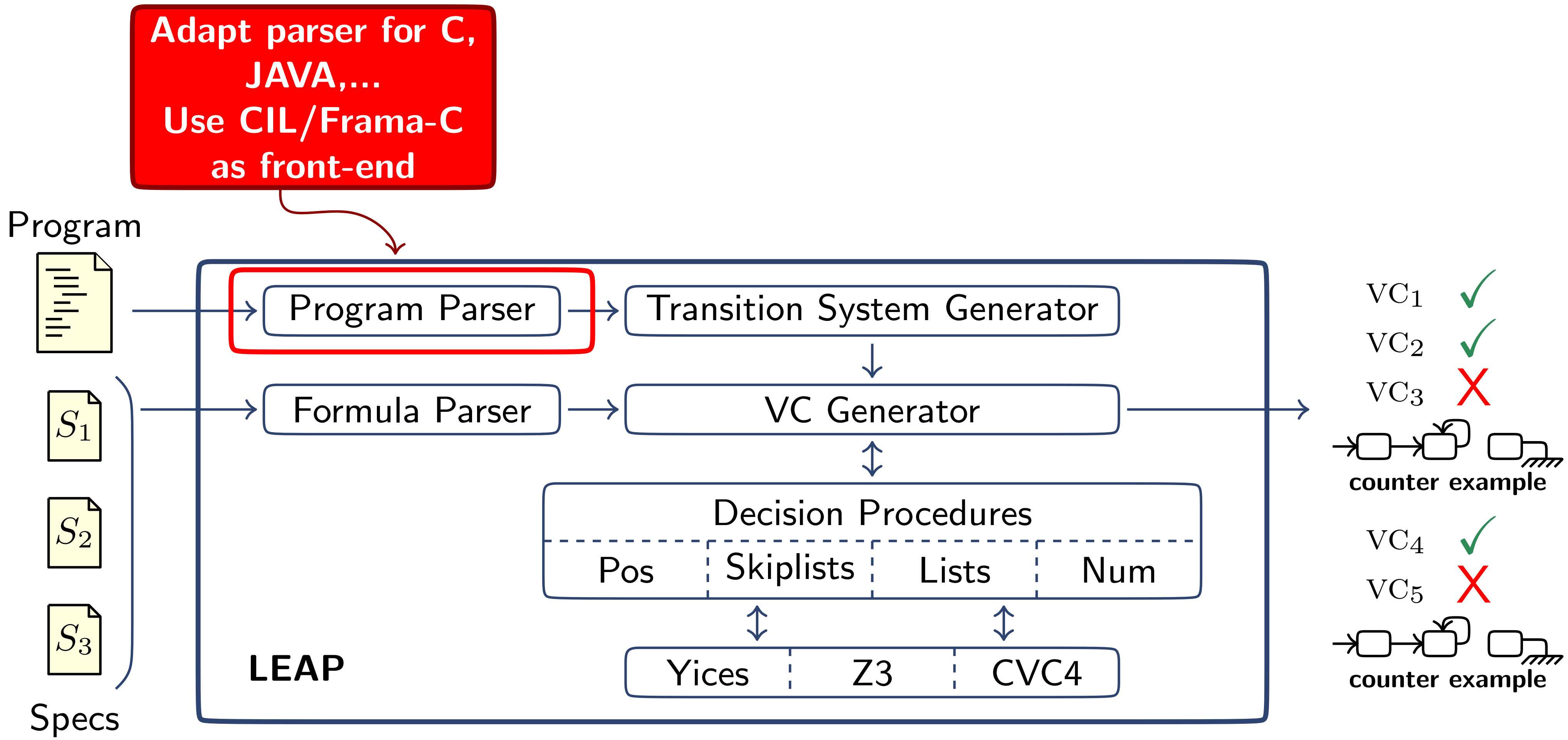
		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	18.17	0.18	0.23
6	disj	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.01	0.01	0.12
7	funSchLinear	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3.04	0.05	0.08
8	funSchInsert	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRemove	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist$_3$	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region$_3$	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next$_3$	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order$_3$	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

Decision procedures perform well...
but still room for improvements

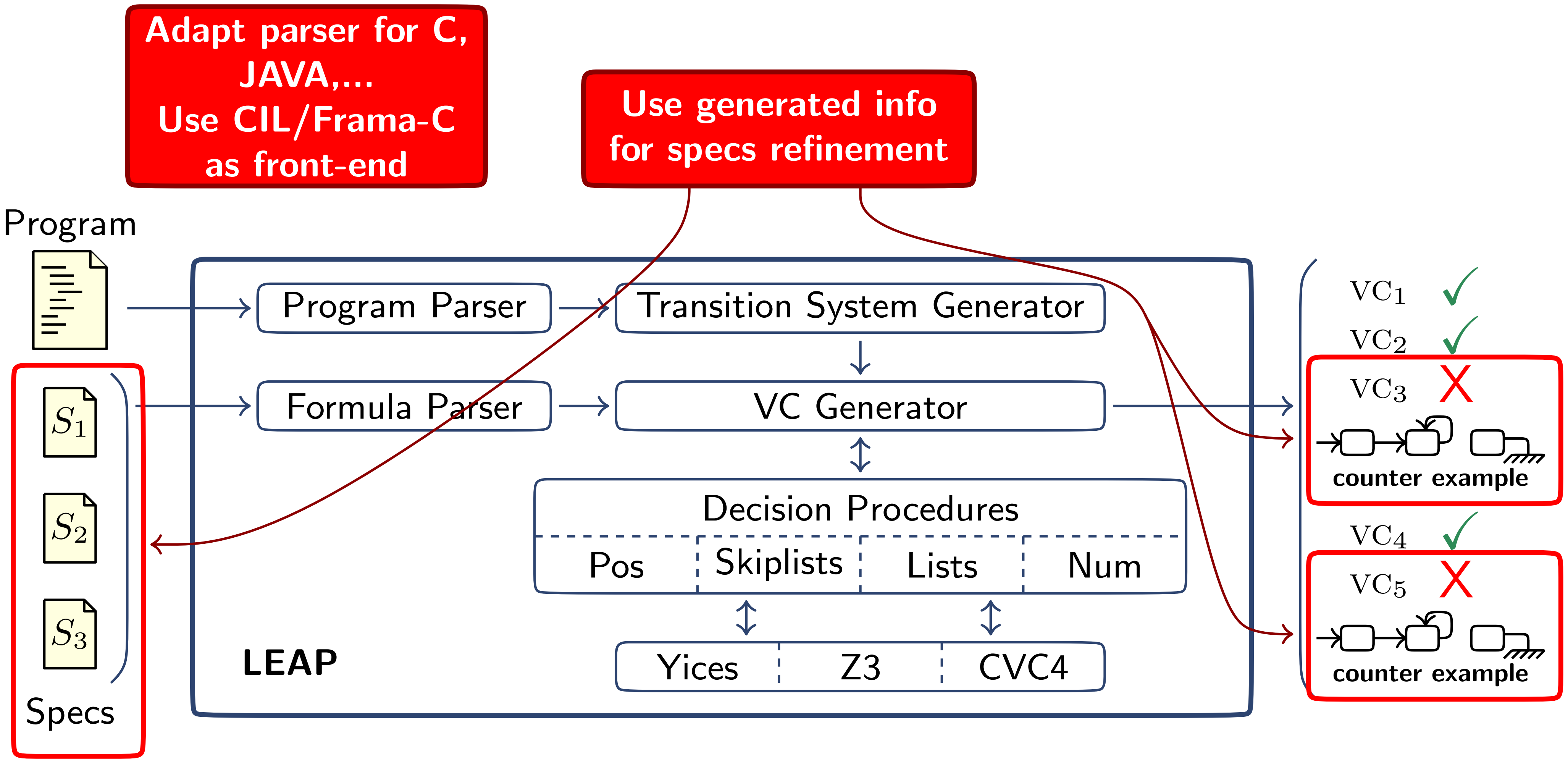
LEAP: Future Directions and Ideas



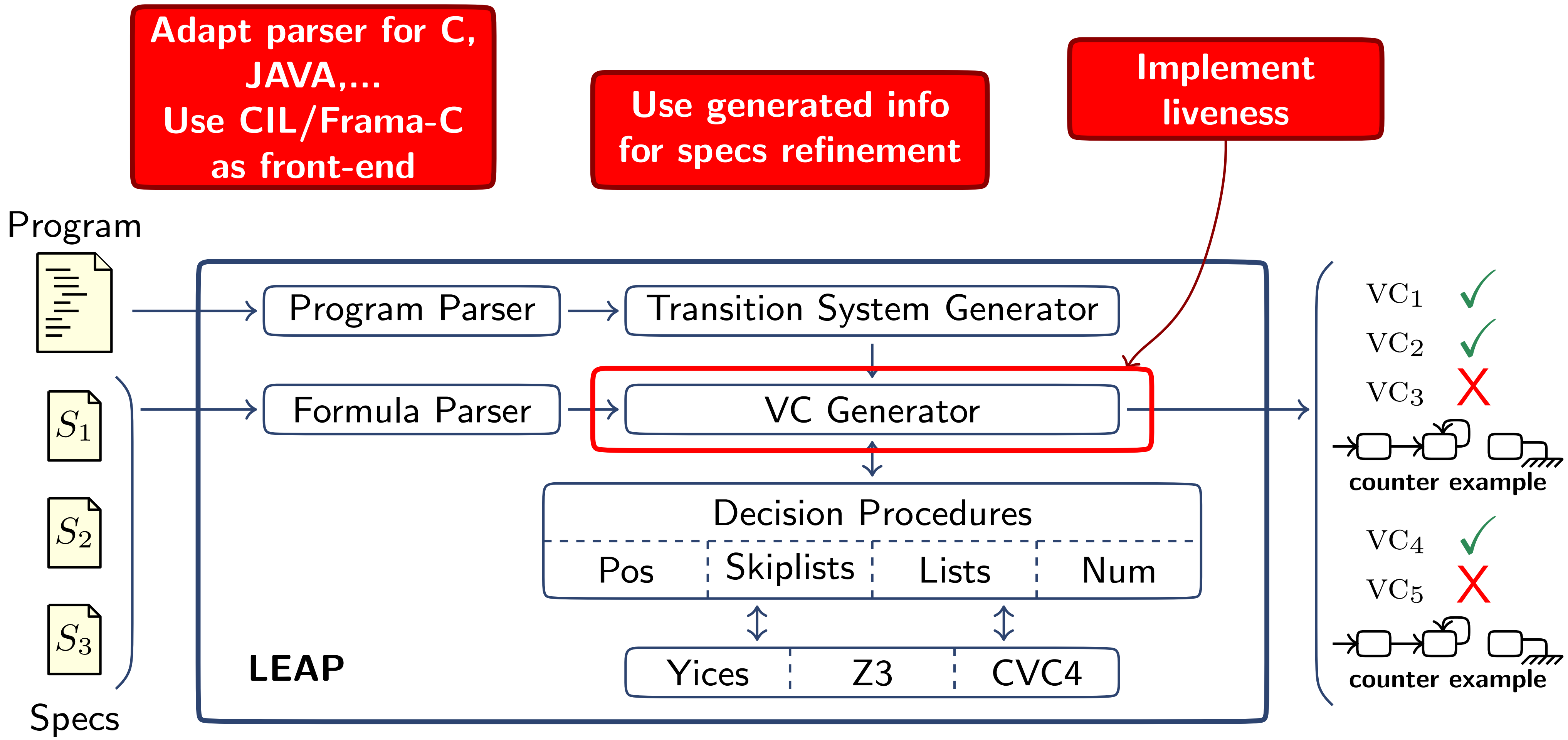
LEAP: Future Directions and Ideas



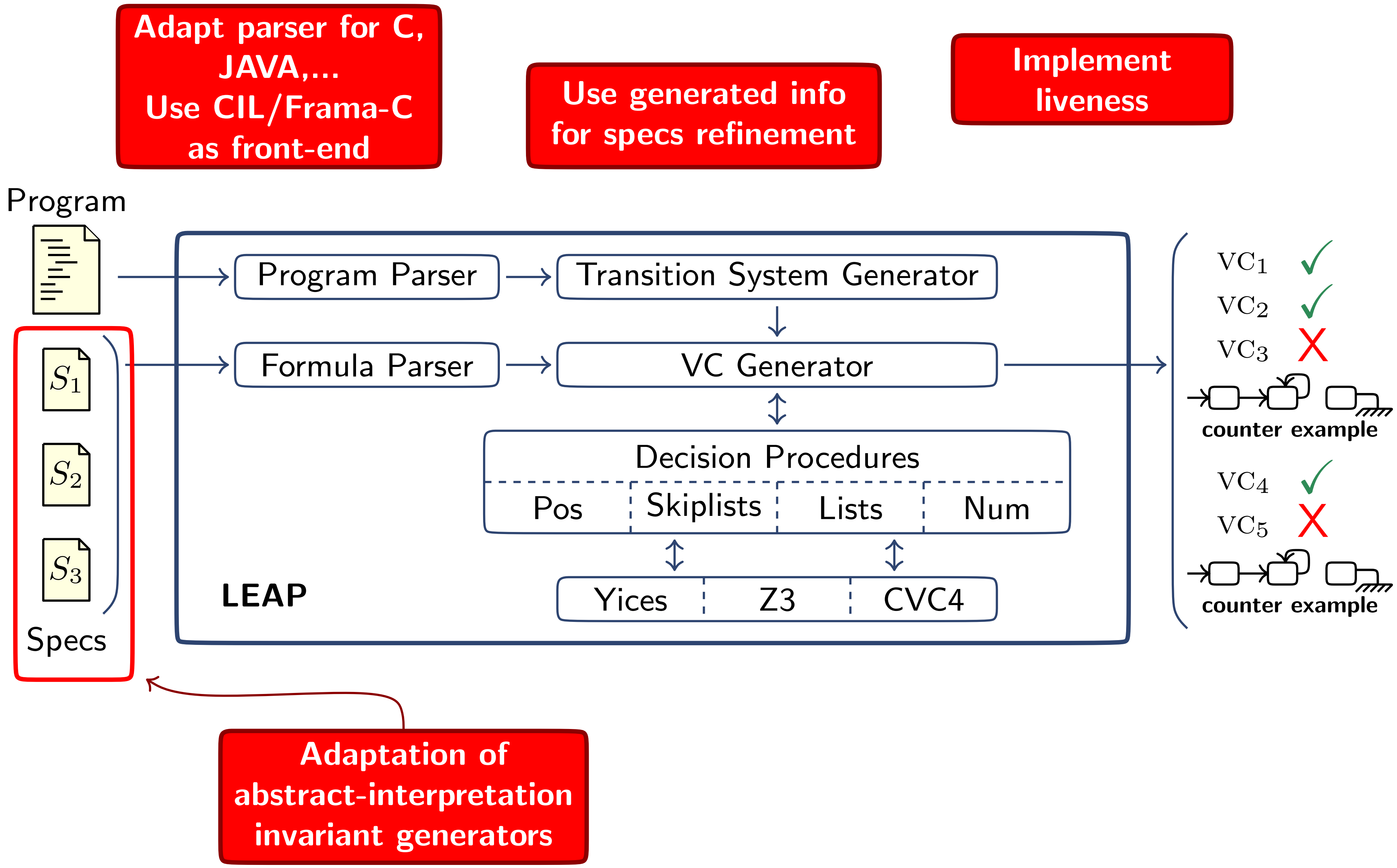
LEAP: Future Directions and Ideas



LEAP: Future Directions and Ideas



LEAP: Future Directions and Ideas

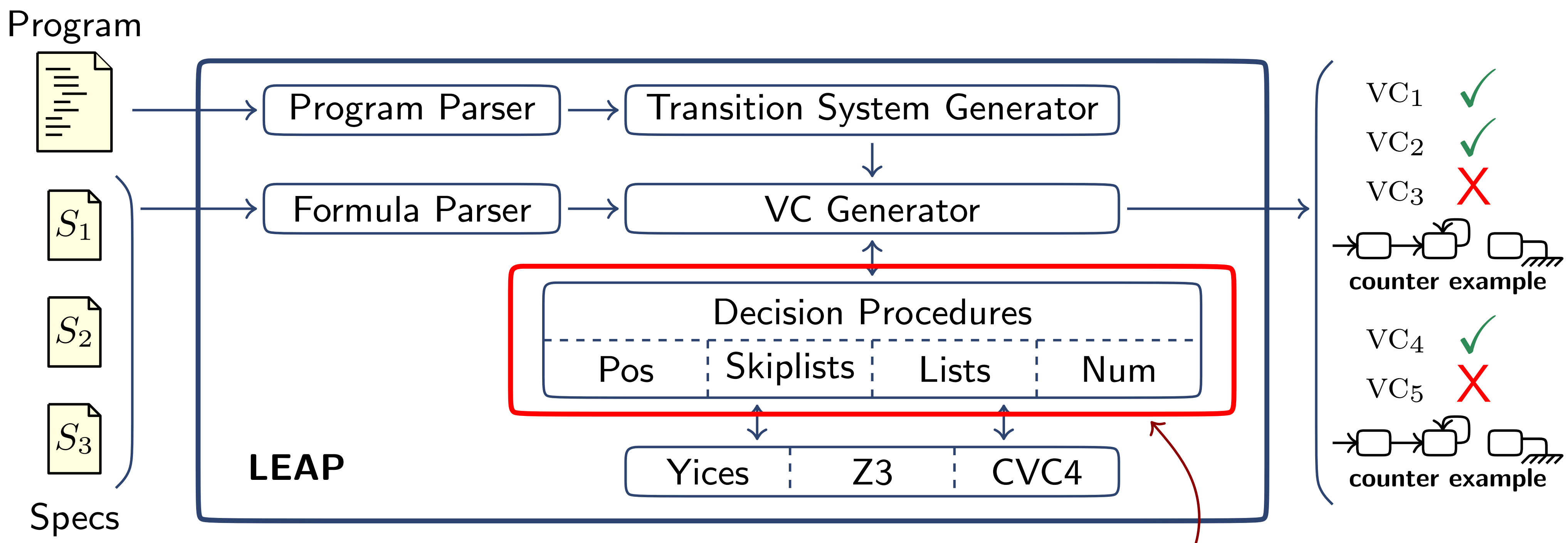


LEAP: Future Directions and Ideas

Adapt parser for C, JAVA,... Use CIL/Frama-C as front-end

Use generated info for specs refinement

Implement liveness



Adaptation of abstract-interpretation invariant generators

More decision procedures: double-linked lists, hashmaps,...

Conclusions

- **Deductive** verification tool for **concurrent datatypes**...
- ... using specialized **decision procedures**
- Aiming the verification of **parametrized symmetric systems**
- **Safety** fully implemented, **liveness** ongoing work

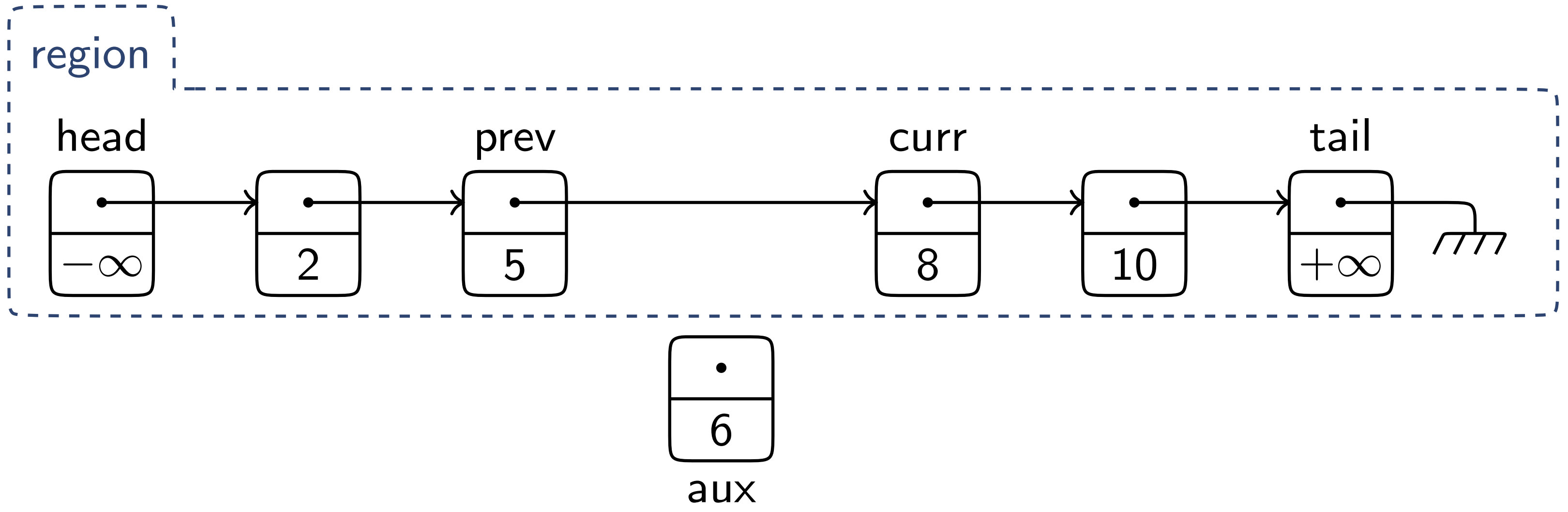
LEAP and examples **available online** at

`software.imdea.org/leap`

(demo on demand)

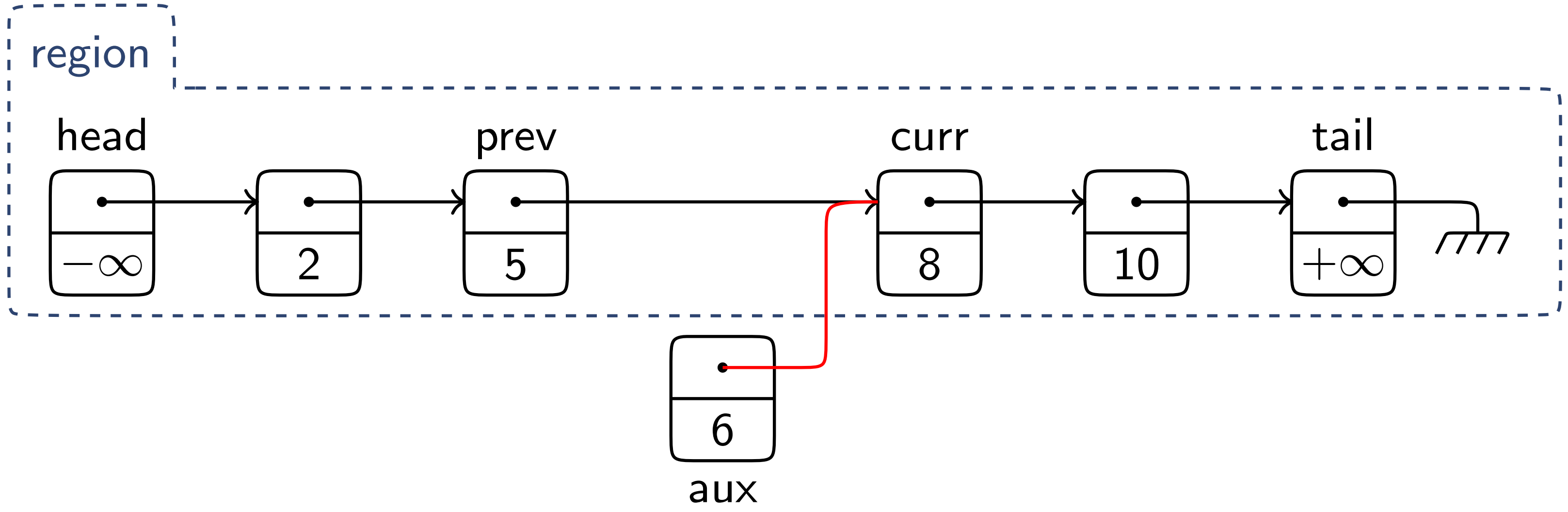
Demo

- **List preservation** concurrent lock-coupling single-linked lists



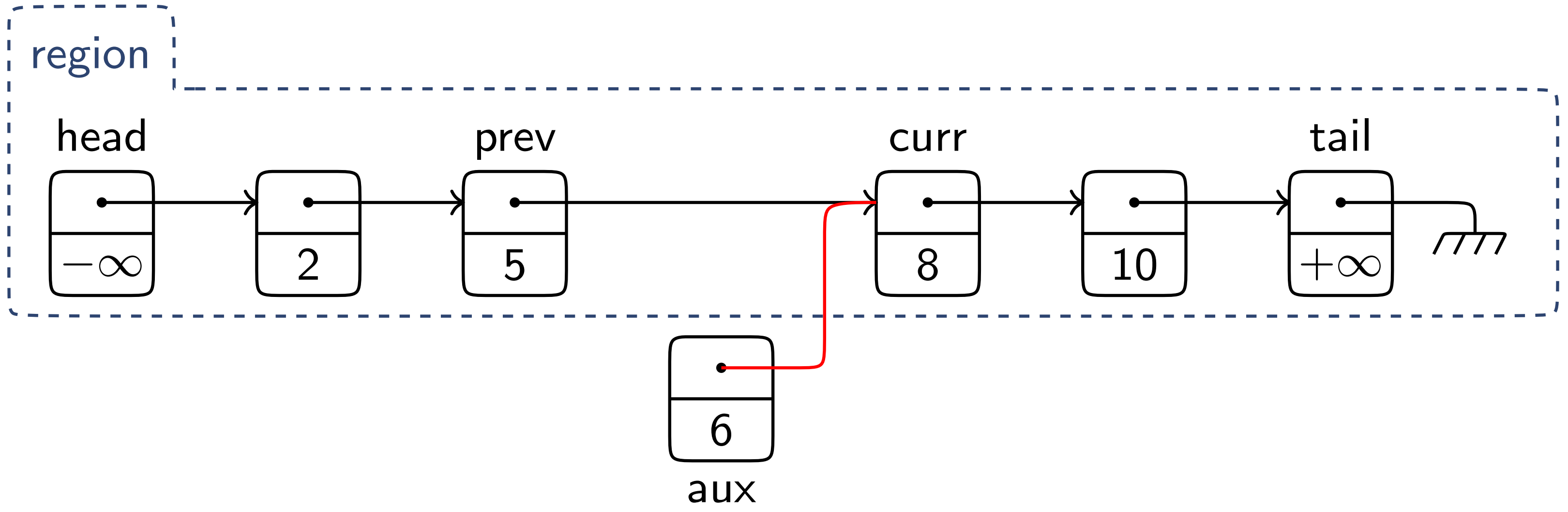
Demo

- **List preservation** concurrent lock-coupling single-linked lists
- Transition 34: `aux->next := curr`

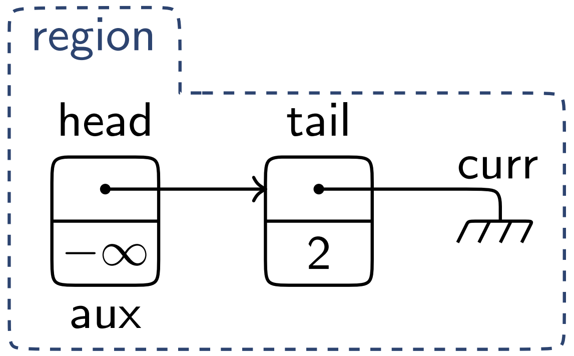


Demo

- **List preservation** concurrent lock-coupling single-linked lists
- Transition 34: `aux->next := curr`

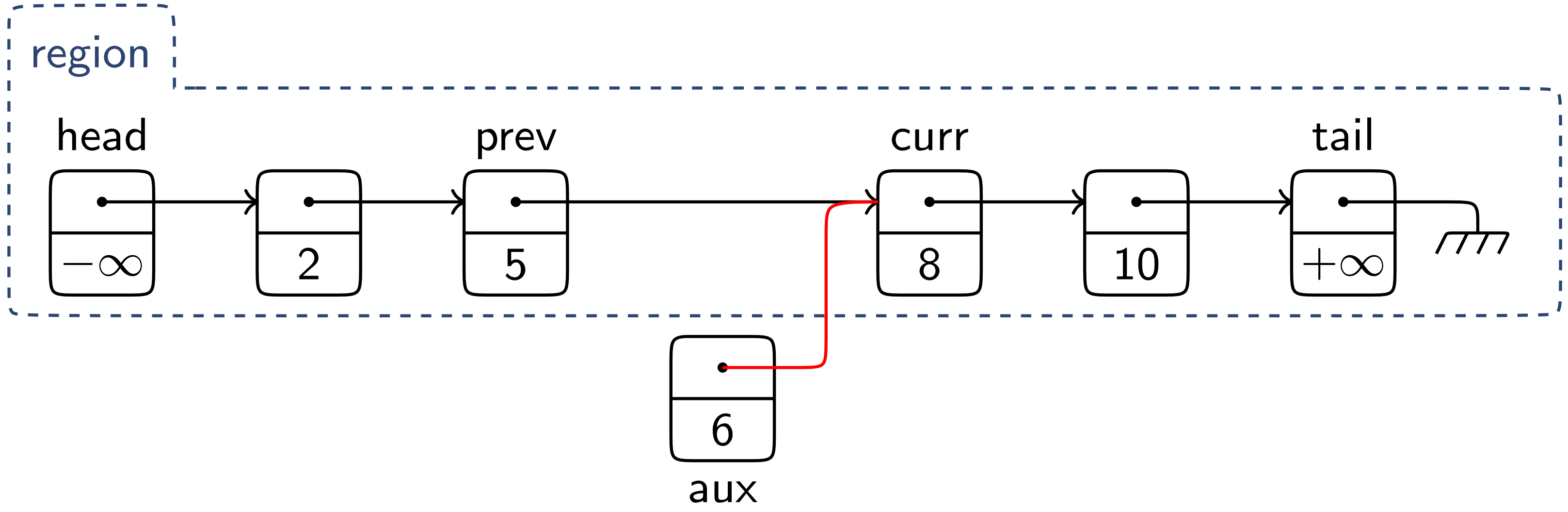


- **Counter example**

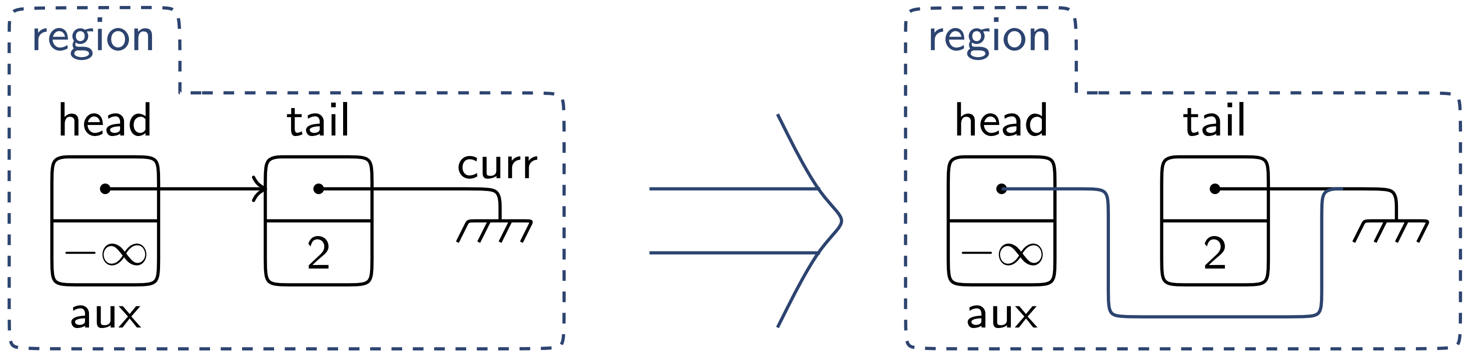


Demo

- **List preservation** concurrent lock-coupling single-linked lists
- Transition 34: `aux->next := curr`



- **Counter example**


region
head
tail
curr
$-\infty$
2
aux
region
head
tail
$-\infty$
2
aux