Assisted Verification of Invariance for Parametrized Systems

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

[1]IMDEA Software Institute, Spain

[2]Spanish Council for Scientific Research (CSIC), Spain

Rich Model Toolkit COST Action Meeting, Haifa, 4 November 2012

Motivation

Motivation

$$S[N]: \quad T_1 \quad T_2 \quad \cdots \quad T_N$$

- Consider a **finite** but **unbounded** set of threads

Motivation

$$S[N] : \quad T_1 \quad \| \quad T_2 \quad \| \quad \cdots \quad \| \quad T_N$$

- Consider a **finite** but **unbounded** set of threads
- Running the **same program** in **paralallel**
- Classical scenarios include:
  - Device drivers
  - Distributed algorithms
  - Concurrent datastructures
  - Robotic swarms
  - Biological Systems

Motivation

$S[N]:$

GOAL : **Prove parametrized invariance** $\varphi(\overline{v})$

$$S[N] \models \varphi(\overline{v}) \qquad \text{for all } N > 1$$

- Consider a **finite** but **unbounded** set of threads
- Running the **same program** in **paralallel**
- Classical scenarios include:
  - Device drivers
  - Distributed algorithms
  - Concurrent datastructures
  - Robotic swarms
  - Biological Systems

Motivating Example: Ticket Mutex Algorithm

Motivating Example: Ticket Mutex Algorithm

global

$Int\ avail := 0$

$Set\langle Int\rangle\ bag := \emptyset$

procedure MUTEXC

$Int\ ticket$

begin

loop

nondet

$\left\langle \begin{array}{l} ticket := avail{+}{+} \\ bag.add(ticket) \end{array} \right\rangle$

await $(bag.min == ticket)$

critical

$bag.remove(ticket)$

end loop

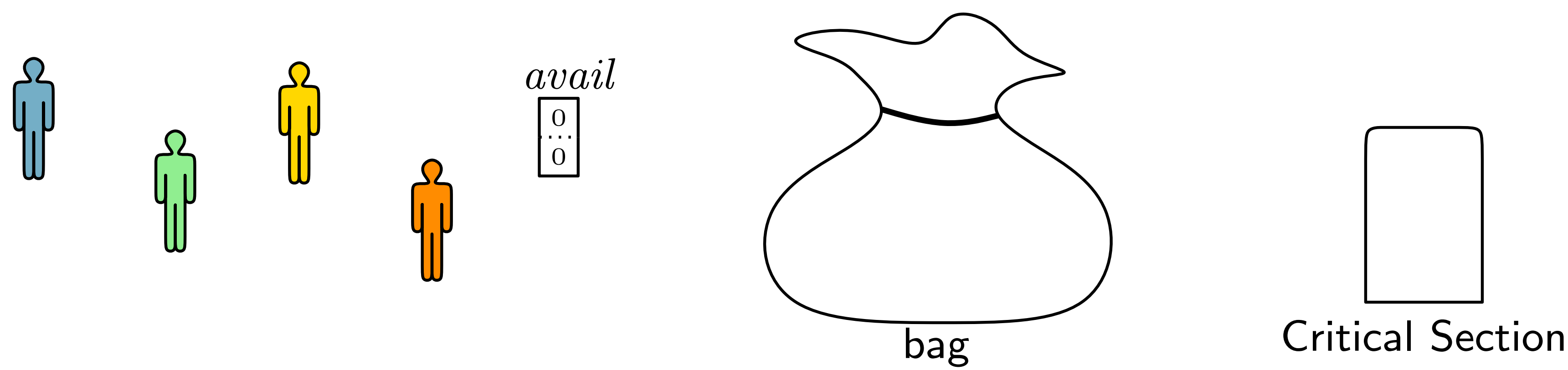
end procedure

Motivating Example: Ticket Mutex Algorithm

```
global
   Int avail := 0
   Set⟨Int⟩ bag := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := avail++
        bag.add(ticket) ⟩
      await (bag.min == ticket)
      critical
      bag.remove(ticket)
end loop
end procedure
```

Motivating Example: Ticket Mutex Algorithm

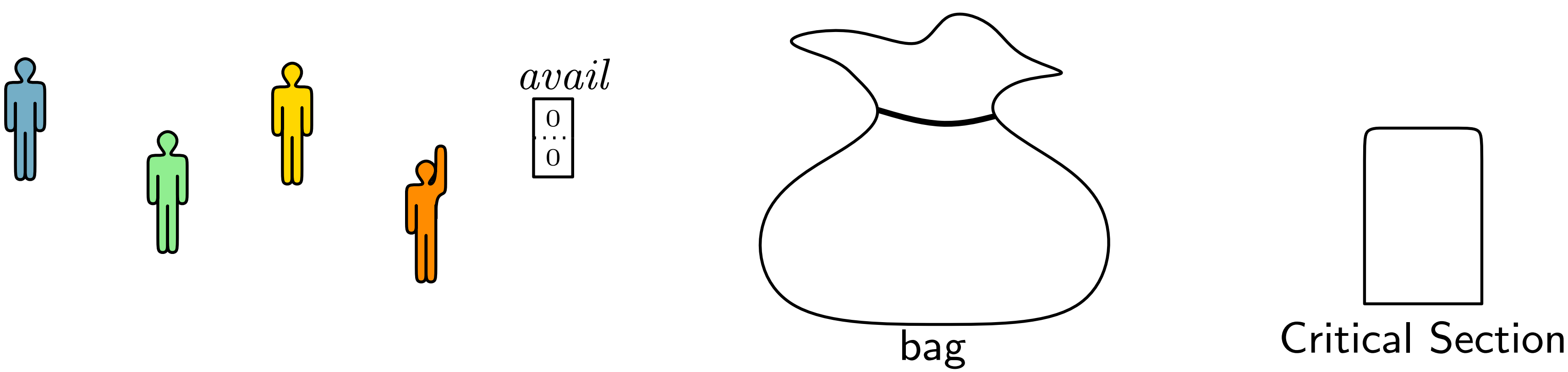
```
global
  Int avail := 0
  Set⟨Int⟩ bag := ∅

procedure MutExc
  Int ticket
begin
loop
    nondet
    ⟨ ticket := avail++
      bag.add(ticket) ⟩
    await (bag.min == ticket)
    critical
    bag.remove(ticket)
end loop
end procedure
```



Motivating Example: Ticket Mutex Algorithm

global
  $Int\ avail := 0$
  $Set\langle Int\rangle\ bag := \emptyset$

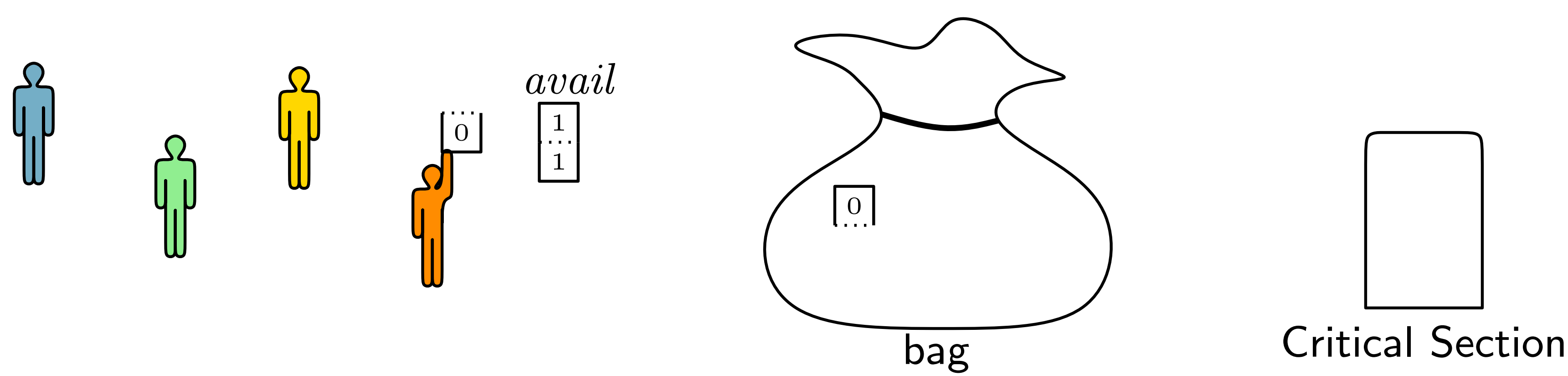
procedure MUTEXC
  $Int\ ticket$
begin
loop
  nondet
    $\left\langle \begin{array}{l} ticket := avail{+}{+} \\ bag.add(ticket) \end{array} \right\rangle$
  await $(bag.min == ticket)$
  critical
  $bag.remove(ticket)$
end loop
end procedure



Motivating Example: Ticket Mutex Algorithm

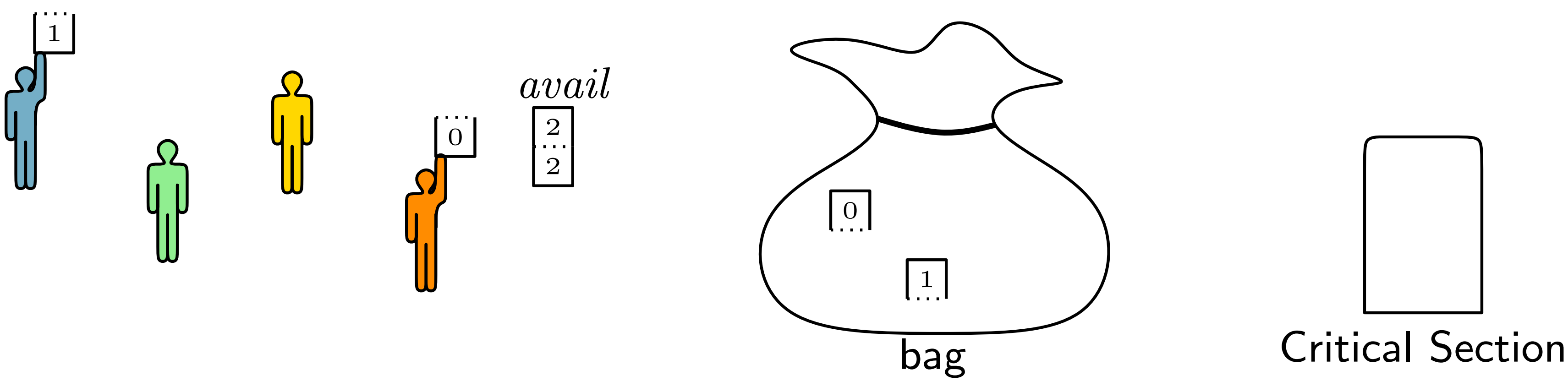
global
  $\mathit{Int}\ \mathit{avail} := 0$
  $\mathit{Set}\langle \mathit{Int} \rangle\ \mathit{bag} := \emptyset$

procedure MUTEXC
  $\mathit{Int}\ \mathit{ticket}$
begin
loop
  nondet
  $\left\langle \begin{array}{l} \mathit{ticket} := \mathit{avail}{+}{+} \\ \mathit{bag}.\mathit{add}(\mathit{ticket}) \end{array} \right\rangle$
  await $(\mathit{bag}.\mathit{min} == \mathit{ticket})$
  critical
  $\mathit{bag}.\mathit{remove}(\mathit{ticket})$
end loop
end procedure



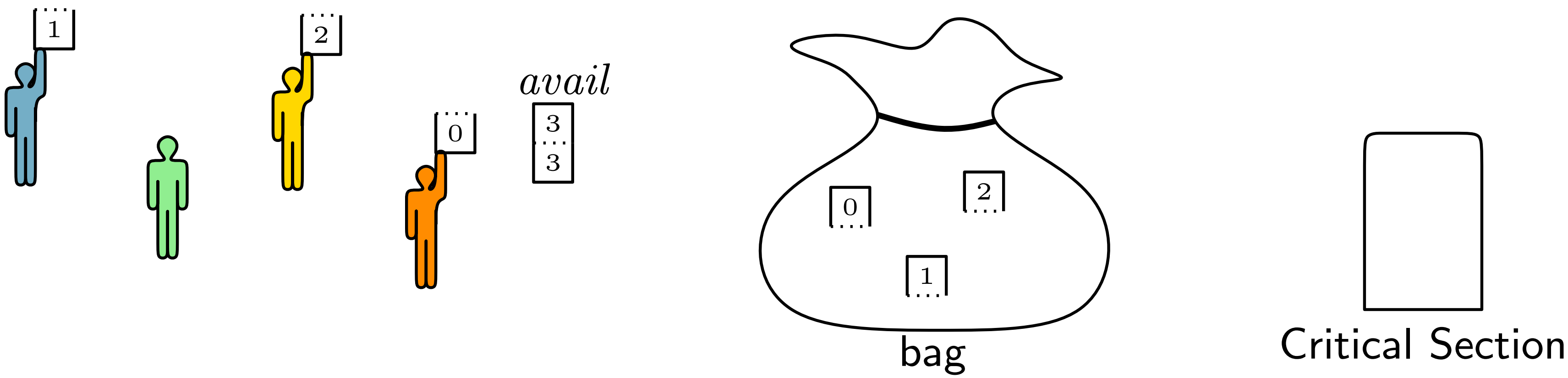
Motivating Example: Ticket Mutex Algorithm

```
global
   Int avail := 0
   Set⟨Int⟩ bag := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := avail++
        bag.add(ticket) ⟩
      await (bag.min == ticket)
      critical
      bag.remove(ticket)
end loop
end procedure
```



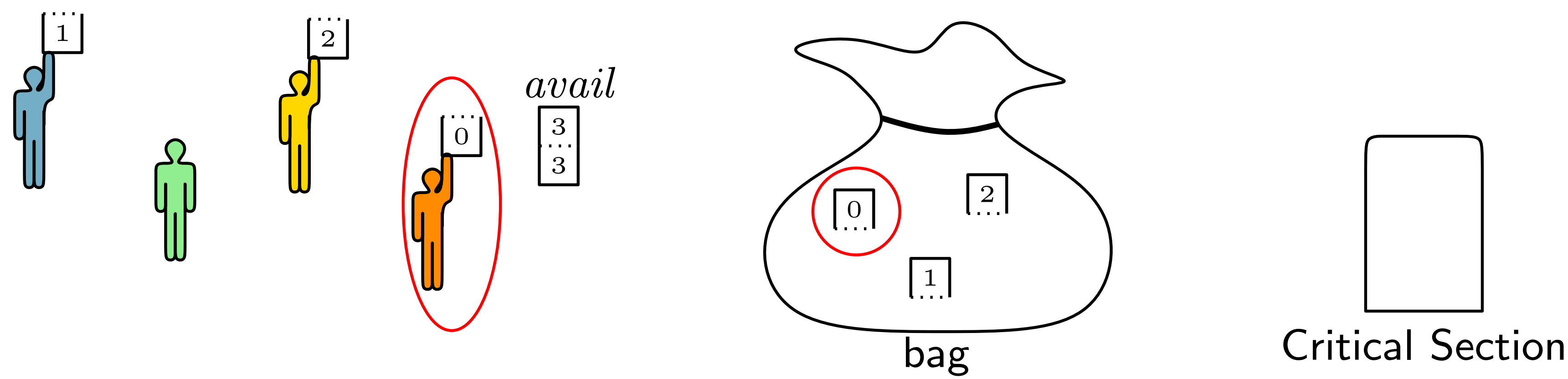
Motivating Example: Ticket Mutex Algorithm

```
global
   Int avail := 0
   Set⟨Int⟩ bag := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := avail ++
        bag.add(ticket) ⟩
      await (bag.min == ticket)
      critical
      bag.remove(ticket)
end loop
end procedure
```



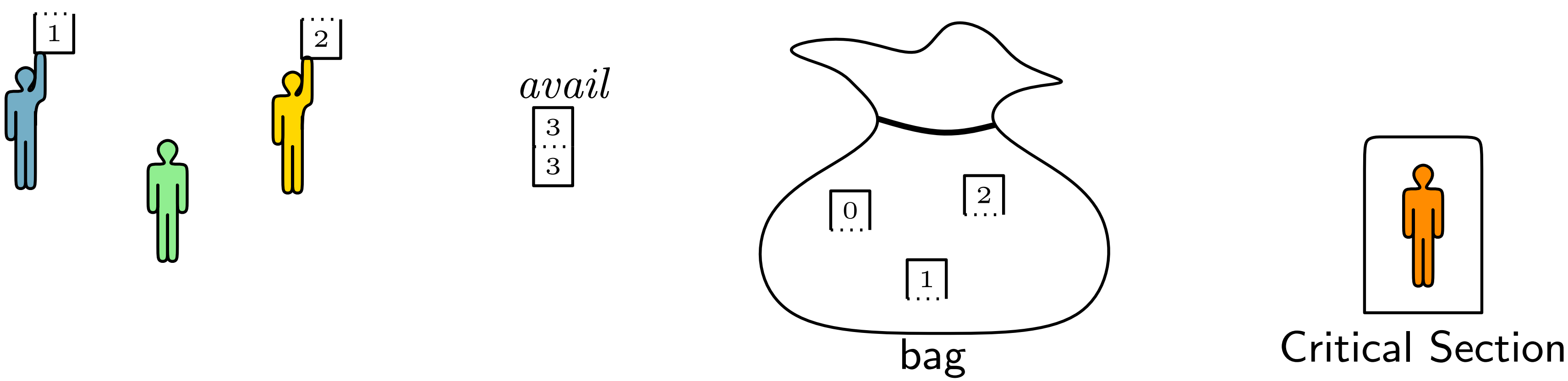
Motivating Example: Ticket Mutex Algorithm

```
global
   Int avail := 0
   Set⟨Int⟩ bag := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := avail++
        bag.add(ticket) ⟩
      await (bag.min == ticket)
      critical
      bag.remove(ticket)
end loop
end procedure
```



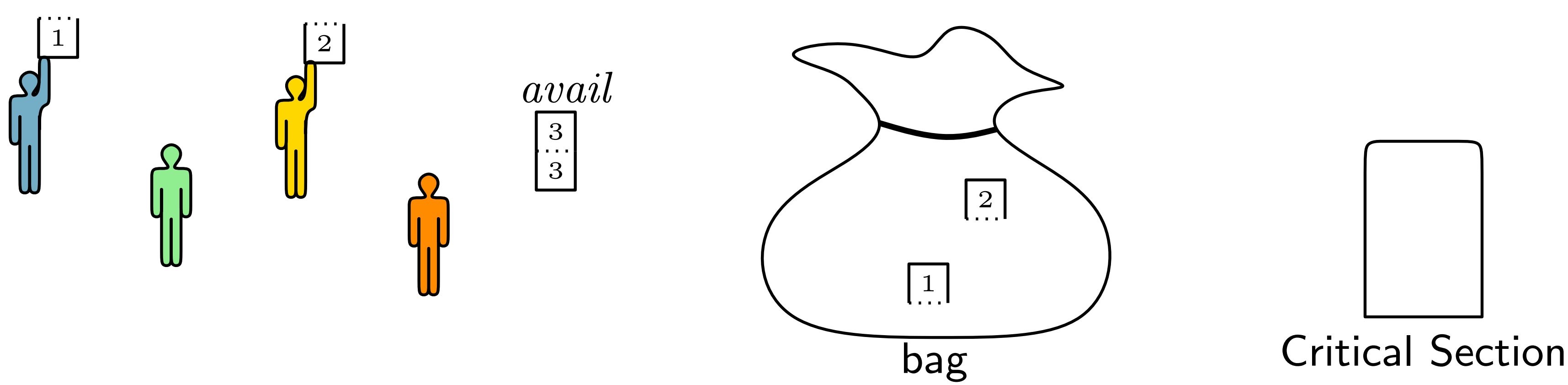
Motivating Example: Ticket Mutex Algorithm

```
global
   Int avail := 0
   Set⟨Int⟩ bag := ∅

procedure MutExc
   Int ticket
begin
loop
      nondet
      ⟨ ticket := avail++
        bag.add(ticket) ⟩
      await (bag.min == ticket)
      critical
      bag.remove(ticket)
end loop
end procedure
```



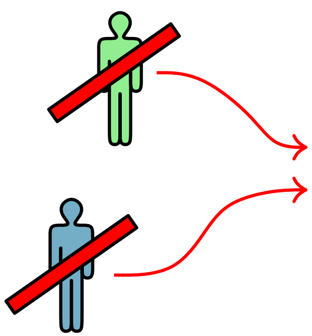
Motivating Example: Ticket Mutex Algorithm

```
global
    Int avail := 0
    Set⟨Int⟩ bag := ∅

procedure MutExc
    Int ticket
begin
loop
    nondet
    ⟨ ticket := avail++
      bag.add(ticket) ⟩
    await (bag.min == ticket)
    critical
    bag.remove(ticket)
end loop
end procedure
```



Motivating Example: Ticket Mutex Algorithm

```
global
    Int avail := 0
    Set⟨Int⟩ bag := ∅

procedure MutExc
    Int ticket
begin
loop
    nondet
    ⟨ ticket := avail++
      bag.add(ticket) ⟩
    await (bag.min == ticket)
    critical
    bag.remove(ticket)
end loop
end procedure
```



$$\mathsf{mutex}(i, j) \mathrel{\hat{=}} \Box\big[i \neq j \rightarrow \neg(critical(i) \wedge critical(j))\big]$$

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

Problem

Unbounded number of VCs

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads
T_1, T_2

- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads
T_1, T_2

- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex		
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ	

2 Threads
T_1, T_2

- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1

(l_1)

General Deductive Verification [Manna-Pnueli '95]

1. $\Theta \rightarrow$ mutex
2. mutex $\wedge\ \tau \rightarrow$ mutex$'$ for each τ

2 Threads
T_1, T_2

- Initiation: **1 VC**

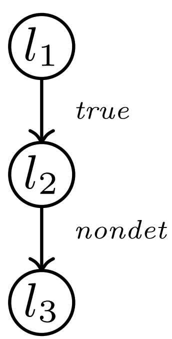
$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification

[Manna-Pnueli '95]

1. $\Theta \rightarrow \text{mutex}$
2. $\text{mutex} \wedge \tau \rightarrow \text{mutex}'$ for each τ

2 Threads T_1, T_2

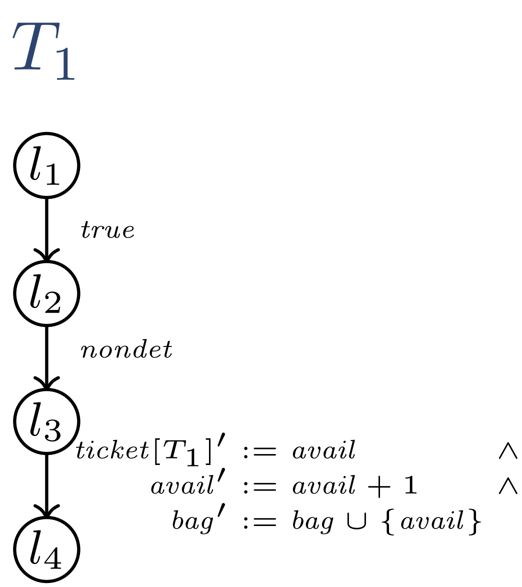
- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:



General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads
T_1, T_2

- Initiation: **1 VC**

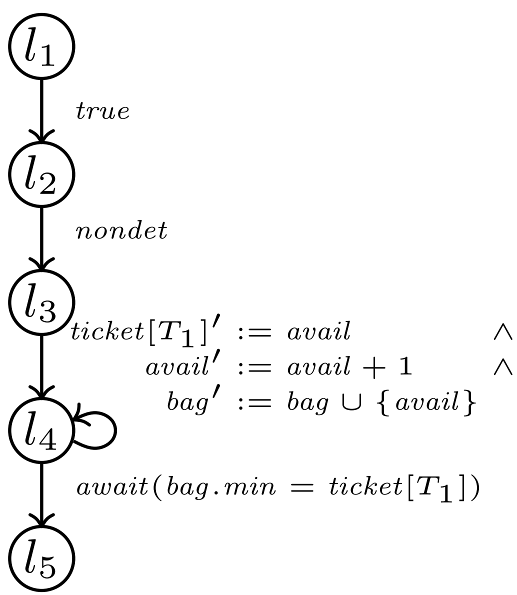
$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification [Manna-Pnueli '95]

1. $\Theta \rightarrow \text{mutex}$
2. $\text{mutex} \wedge \tau \rightarrow \text{mutex}'$ for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

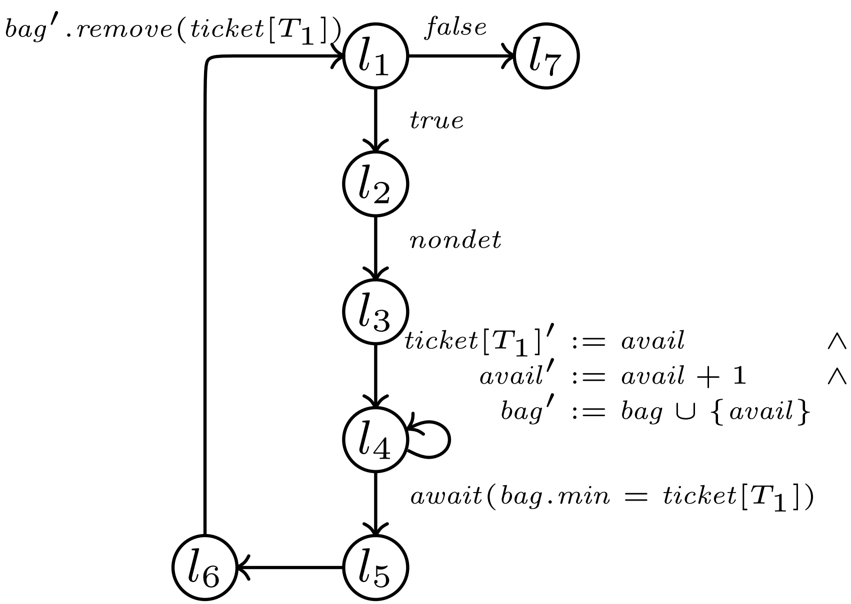
$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification [Manna-Pnueli '95]

$$
\begin{array}{llcll}
1. & \Theta & \rightarrow & \text{mutex} & \\
2. & \text{mutex} \wedge \tau & \rightarrow & \text{mutex}' & \text{for each } \tau
\end{array}
$$

2 Threads T_1, T_2

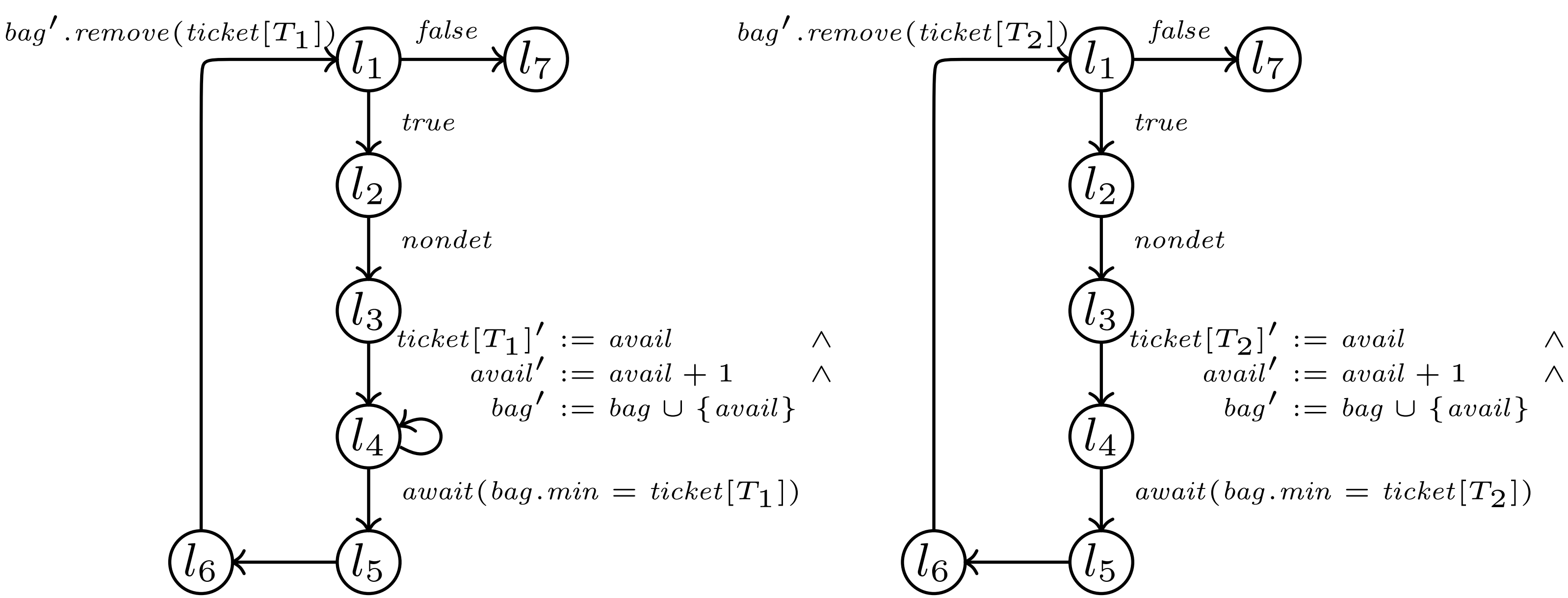
- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$ $\quad$ $\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution: **18 VC**

$T_1 \quad \| \quad T_2$



General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

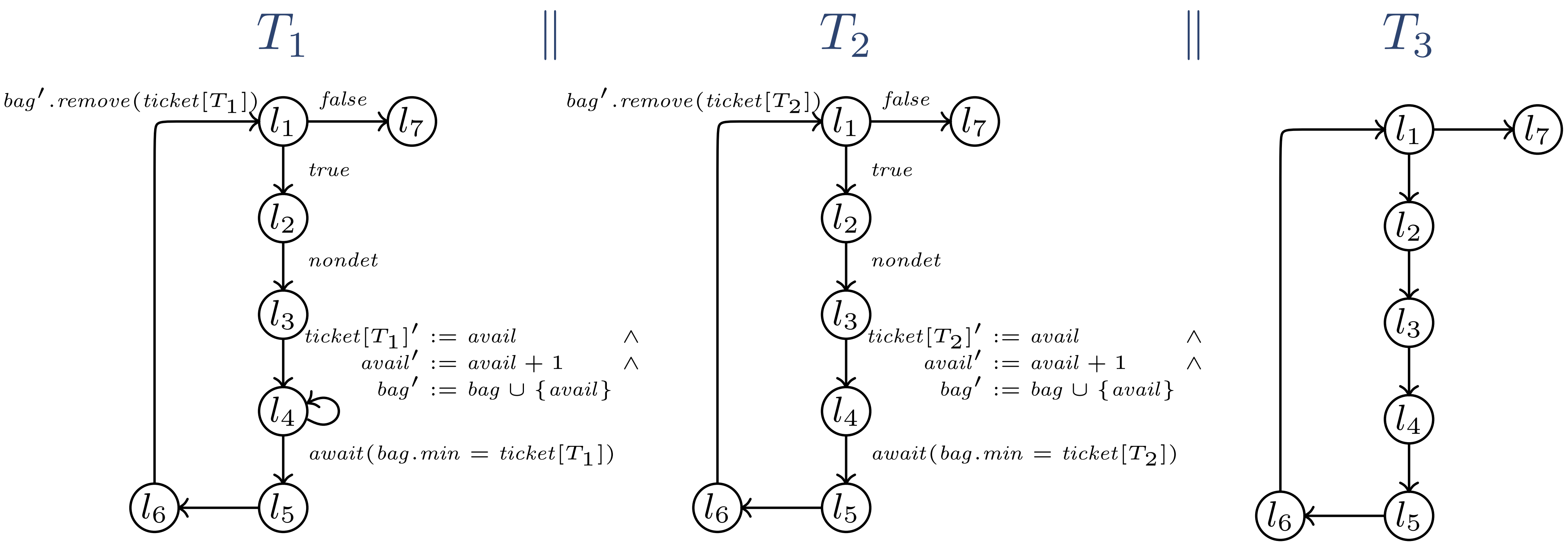
- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution: **18 VC**



General Deductive Verification [Manna-Pnueli '95]

$$\begin{array}{llcll} 1. & \Theta & \rightarrow & \text{mutex} & \\ 2. & \text{mutex} \wedge \tau & \rightarrow & \text{mutex}' & \text{for each } \tau \end{array}$$

2 Threads T_1, T_2

- Initiation: **1 VC**

$\Theta_G : avail = 0 \wedge bag = \emptyset$

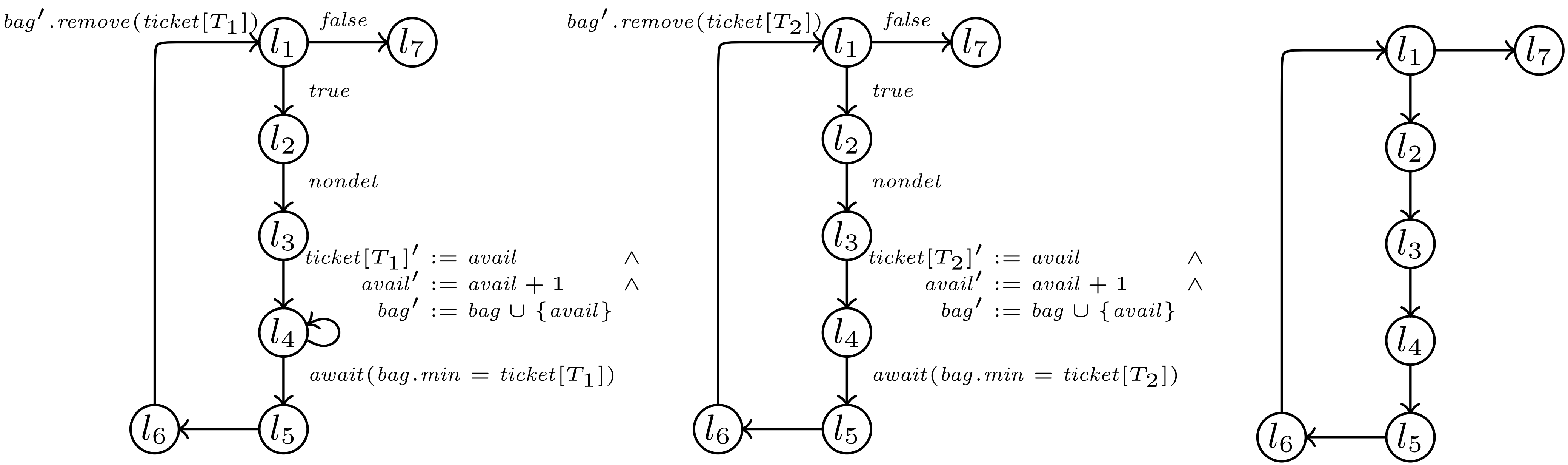
$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

$\Theta_{T_3} : ticket[T_3] = 0 \wedge pc[T_3] = 1$

- Consecution: ~~**18 VC**~~ **27 VC**

$T_1 \quad \| \quad T_2 \quad \| \quad T_3$



Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

$$
\begin{array}{lrcll}
(\;\mathsf{I}\;) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & &
\end{array}
$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$: **Initiation**

$$
\begin{array}{llcll}
(\ \mathsf{I}\ ) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & \Box \varphi & & &
\end{array}
$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

Initiation

Self-consecution

$$
\begin{array}{llcll}
(\ \mathsf{I}\ ) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

$$
\begin{array}{llcll}
(\ \mathsf{I}\ ) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & \Box \varphi & & &
\end{array}
$$

Initiation (I)

Self-consecution (SC)

Other-consecution (OC)

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

$$
\begin{array}{lrcll}
(\text{ I }) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\text{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\text{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & &
\end{array}
$$

Initiation

Self-consecution

Other-consecution

- For our example: mutex(i, j)

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

$$\frac{\begin{array}{llcll} (\text{ I }) & \Theta(\overline{v}) & \rightarrow & \varphi & \\ (\text{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\ (\text{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \end{array}}{\Box \varphi}$$

Initiation (I), **Self-consecution** (SC), **Other-consecution** (OC)

- For our example: $\text{mutex}(i, j)$ **#VC : 1**

$$(\text{ I }) \quad \Theta(i, j) \rightarrow \text{mutex}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$:

Initiation

Self-consecution

Other-consecution

$$
\begin{array}{llcll}
(\text{ I }) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\text{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\text{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & \Box\varphi & & &
\end{array}
$$

- For our example: $\text{mutex}(i, j)$ **#VC : 1 + 18**

$$
\begin{array}{llcll}
(\text{ I }) & \Theta(i, j) & \rightarrow & \text{mutex} & \\
(\text{SC}) & \text{mutex} \wedge \tau^{(i)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\
 & \text{mutex} \wedge \tau^{(j)} & \rightarrow & \text{mutex}' & \text{forall } \tau
\end{array}
$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$: **Initiation** **Self-consecution** **Other-consecution**

$$\frac{\begin{array}{llcll} (\ \mathsf{I}\ ) & \Theta(\overline{v}) & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\ (\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \end{array}}{\Box \varphi}$$

- For our example: $\mathsf{mutex}(i, j)$ **#VC : 1 + 18 + 9 = 28**

$$\begin{array}{llcll} (\ \mathsf{I}\ ) & \Theta(i, j) & \rightarrow & \mathsf{mutex} & \\ (\mathsf{SC}) & \mathsf{mutex} \wedge \tau^{(i)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau \\ & \mathsf{mutex} \wedge \tau^{(j)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau \\ (\mathsf{OC}) & \mathsf{mutex} \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau, \text{fresh } k \end{array}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(\overline{v})$: **Initiation**

Self-consecution

$$
\begin{array}{llcll}
(\ \mathsf{I}\ ) & \Theta(\overline{v}) & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{forall } i \in \overline{v} \\
(\mathsf{OC}) & \varphi \wedge \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & &
\end{array}
$$

Other-consecution

Independently on #threads in the system

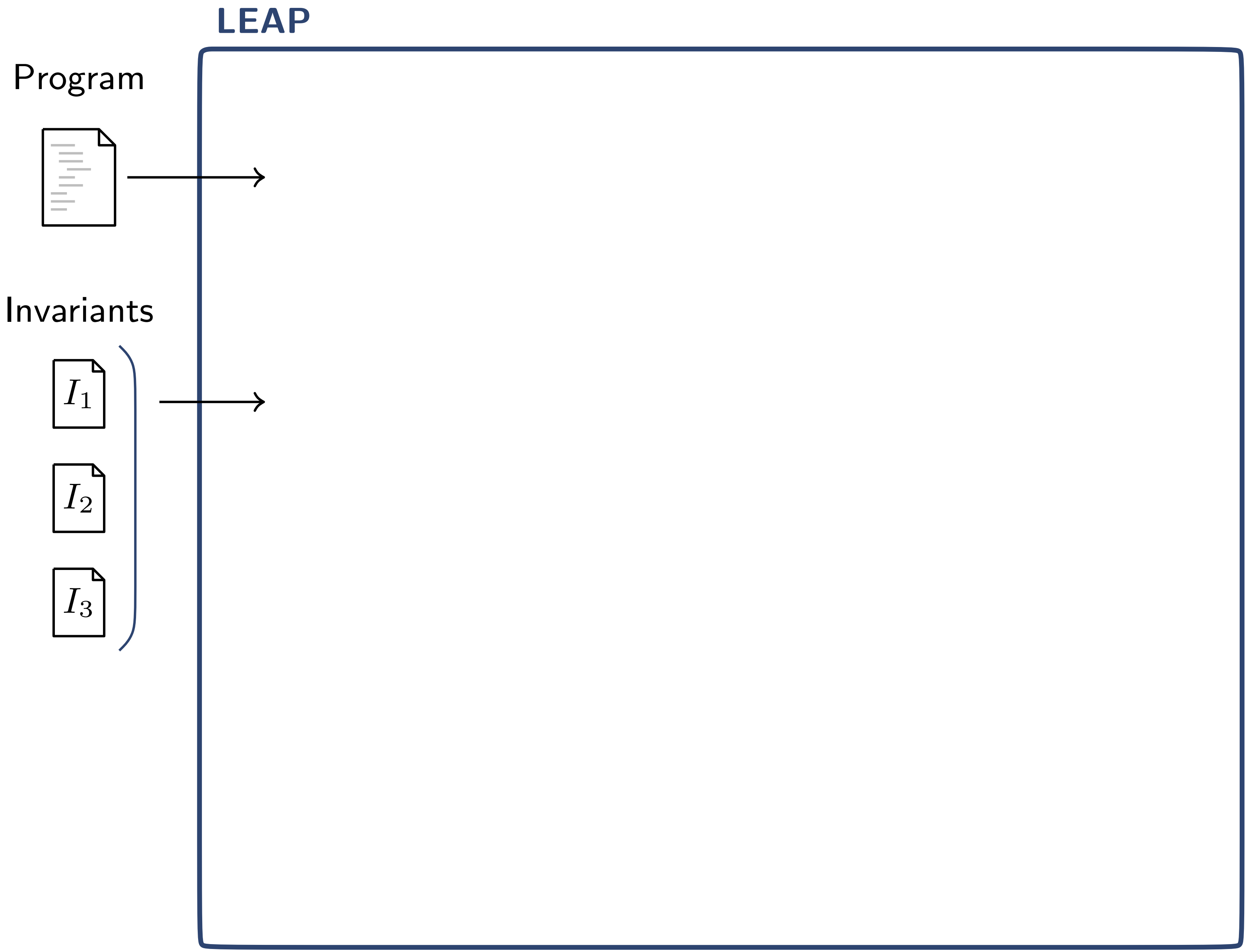
- For our example: $\mathsf{mutex}(i,j)$ **#VC : 1 + 18 + 9 = 28**

$$
\begin{array}{llcll}
(\ \mathsf{I}\ ) & \Theta(i,j) & \rightarrow & \mathsf{mutex} & \\
(\mathsf{SC}) & \mathsf{mutex} \wedge \tau^{(i)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau \\
 & \mathsf{mutex} \wedge \tau^{(j)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau \\
(\mathsf{OC}) & \mathsf{mutex} \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} & \rightarrow & \mathsf{mutex}' & \text{forall } \tau, \text{fresh } k
\end{array}
$$

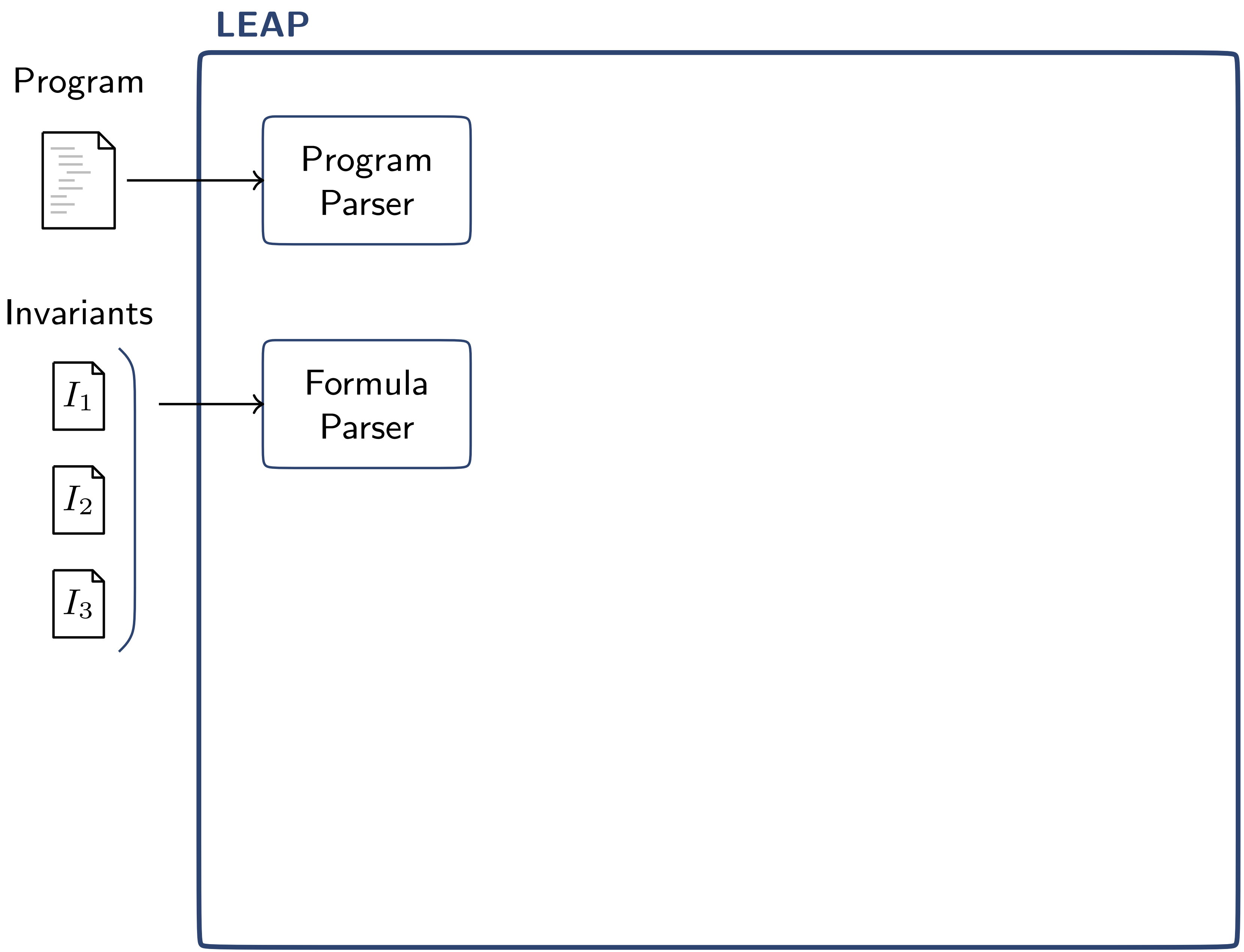
LEAP: A Temporal Deductive Prover

LEAP

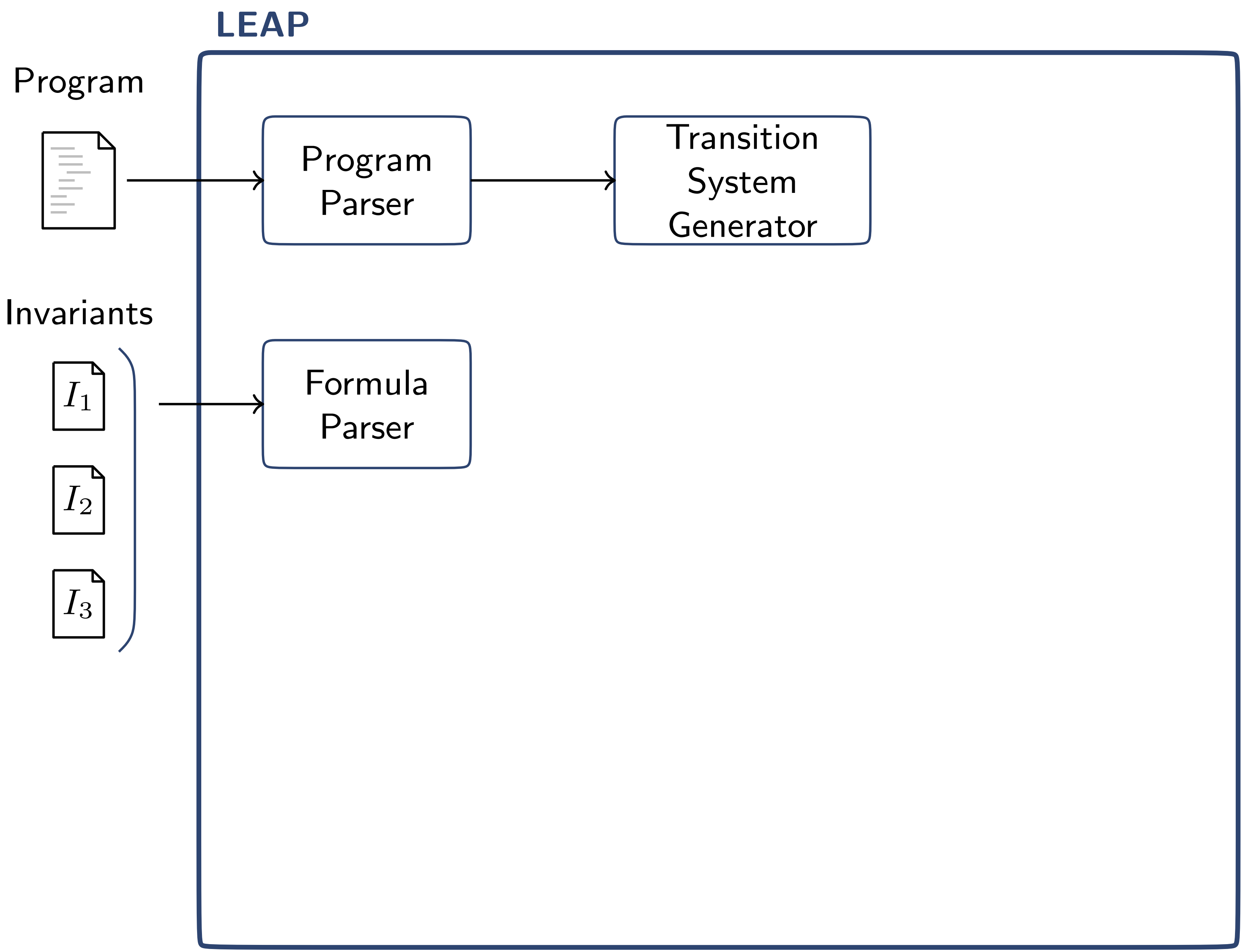
LEAP: A Temporal Deductive Prover



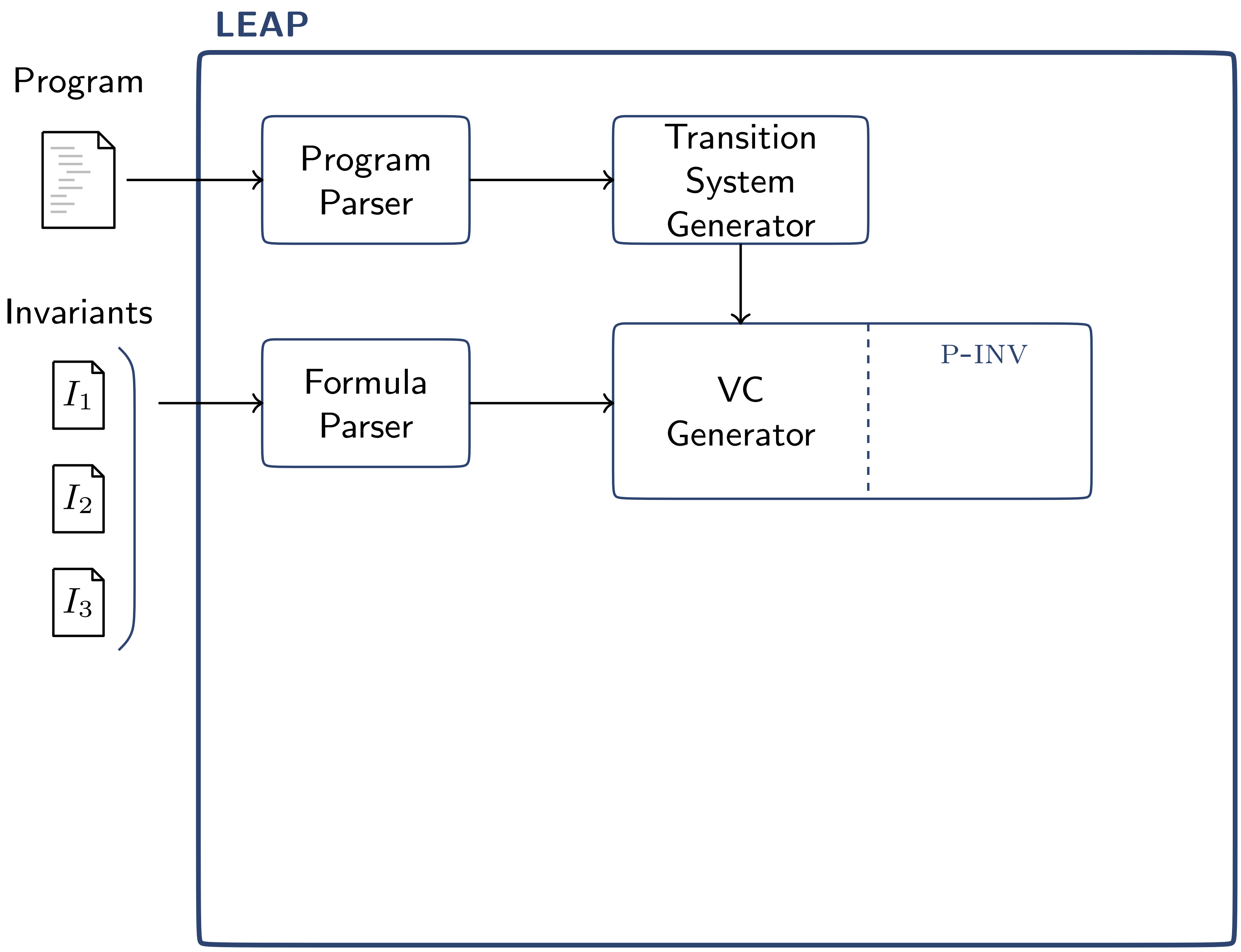
LEAP: A Temporal Deductive Prover



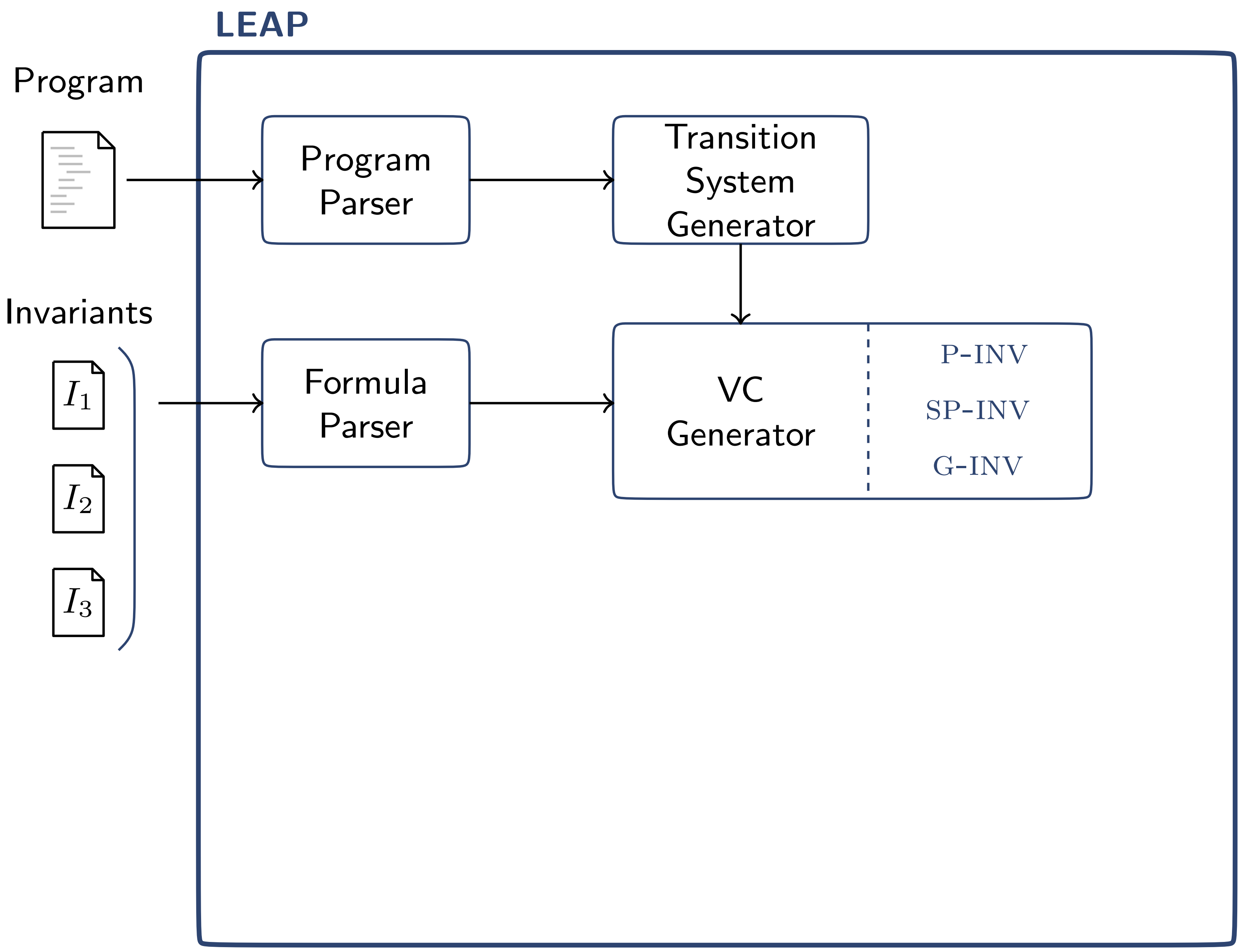
LEAP: A Temporal Deductive Prover



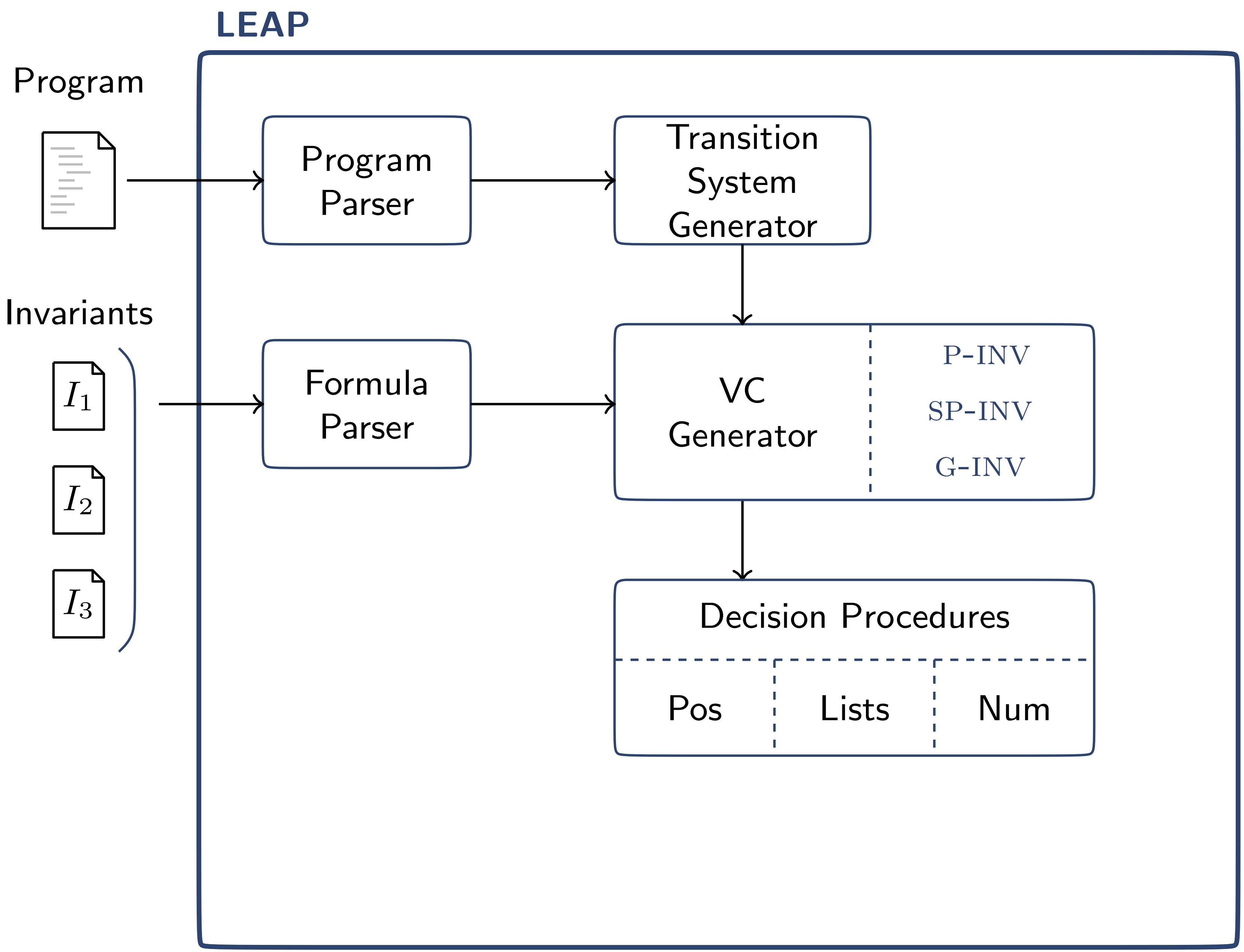
LEAP: A Temporal Deductive Prover



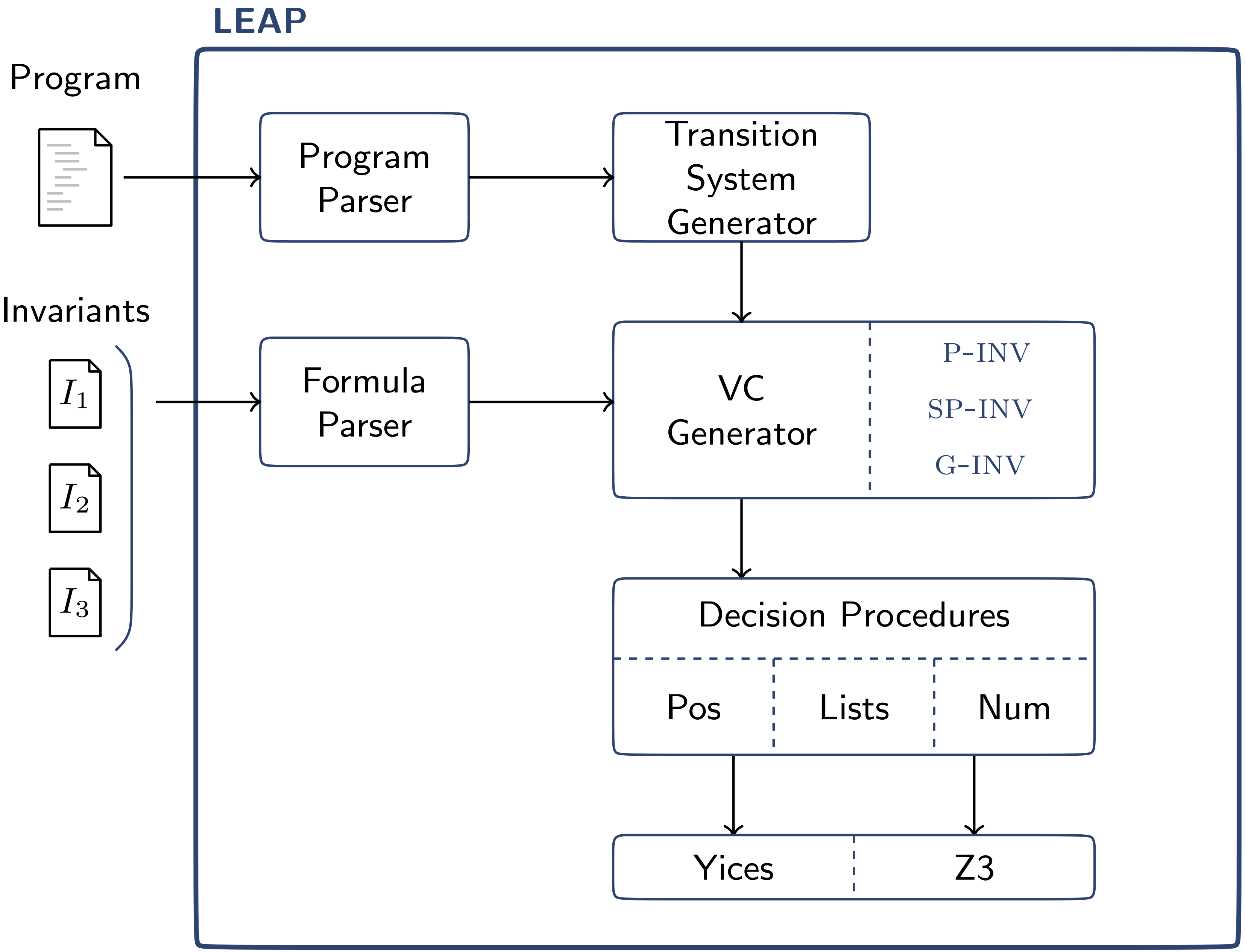
LEAP: A Temporal Deductive Prover



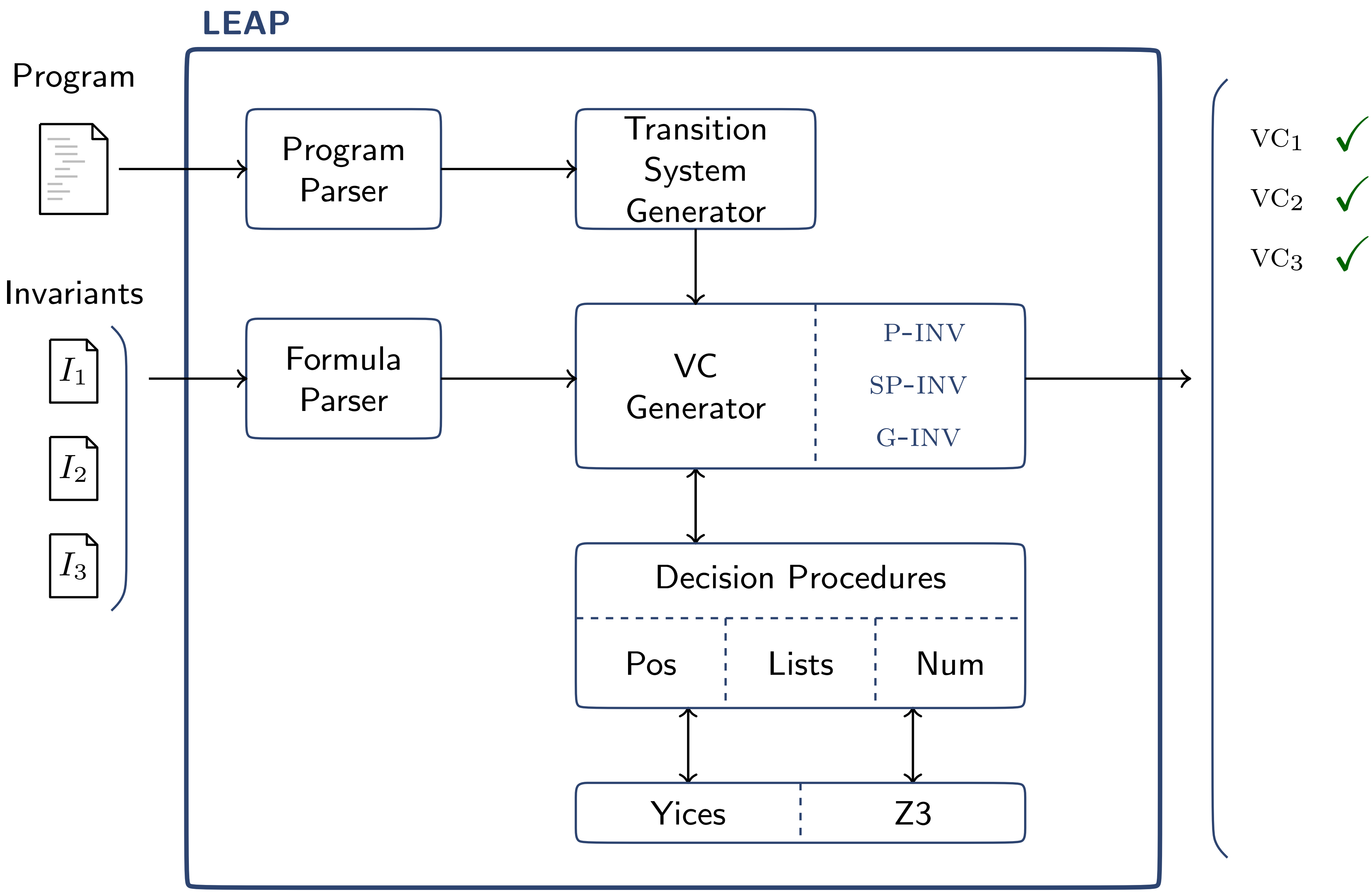
LEAP: A Temporal Deductive Prover



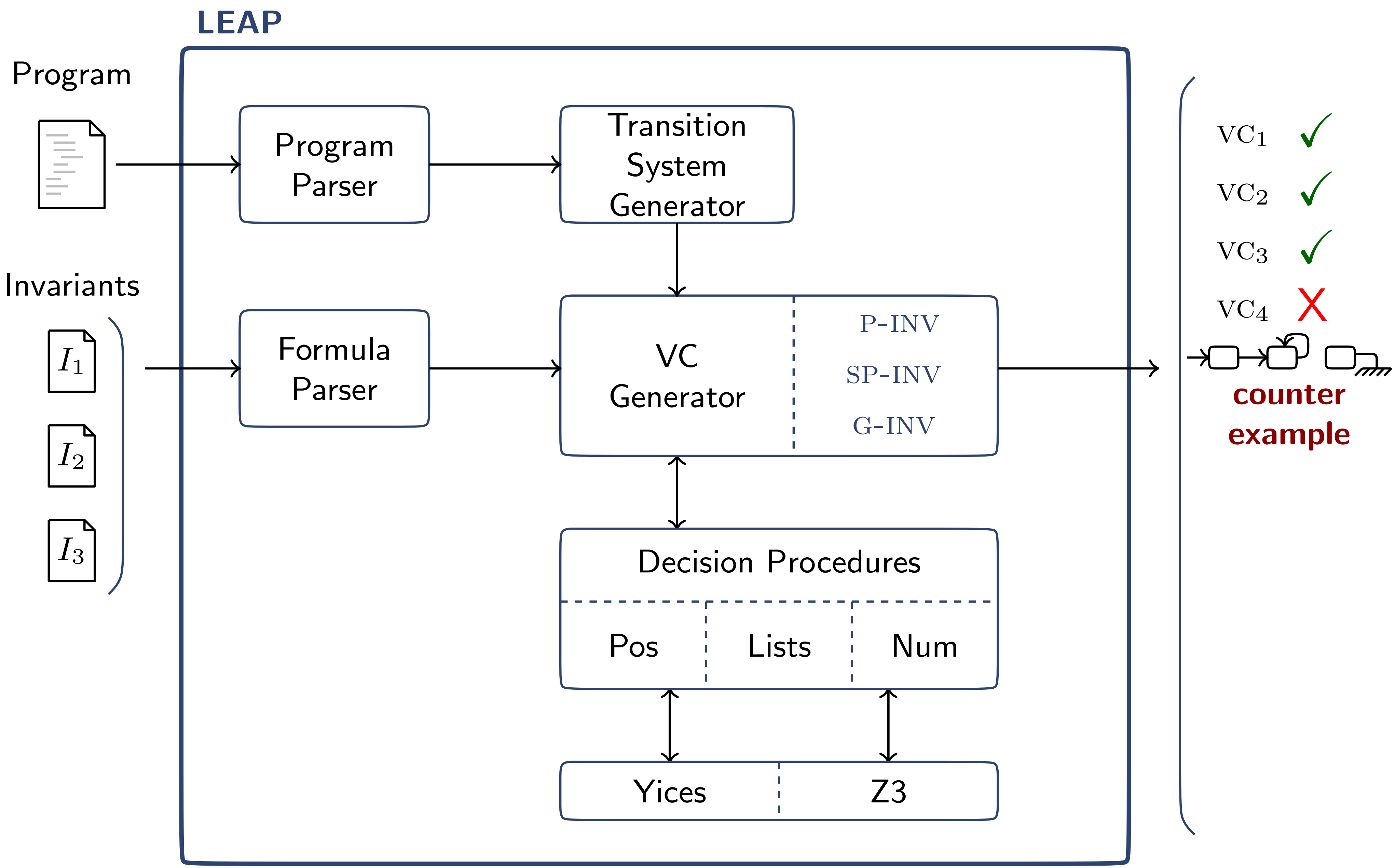
LEAP: A Temporal Deductive Prover



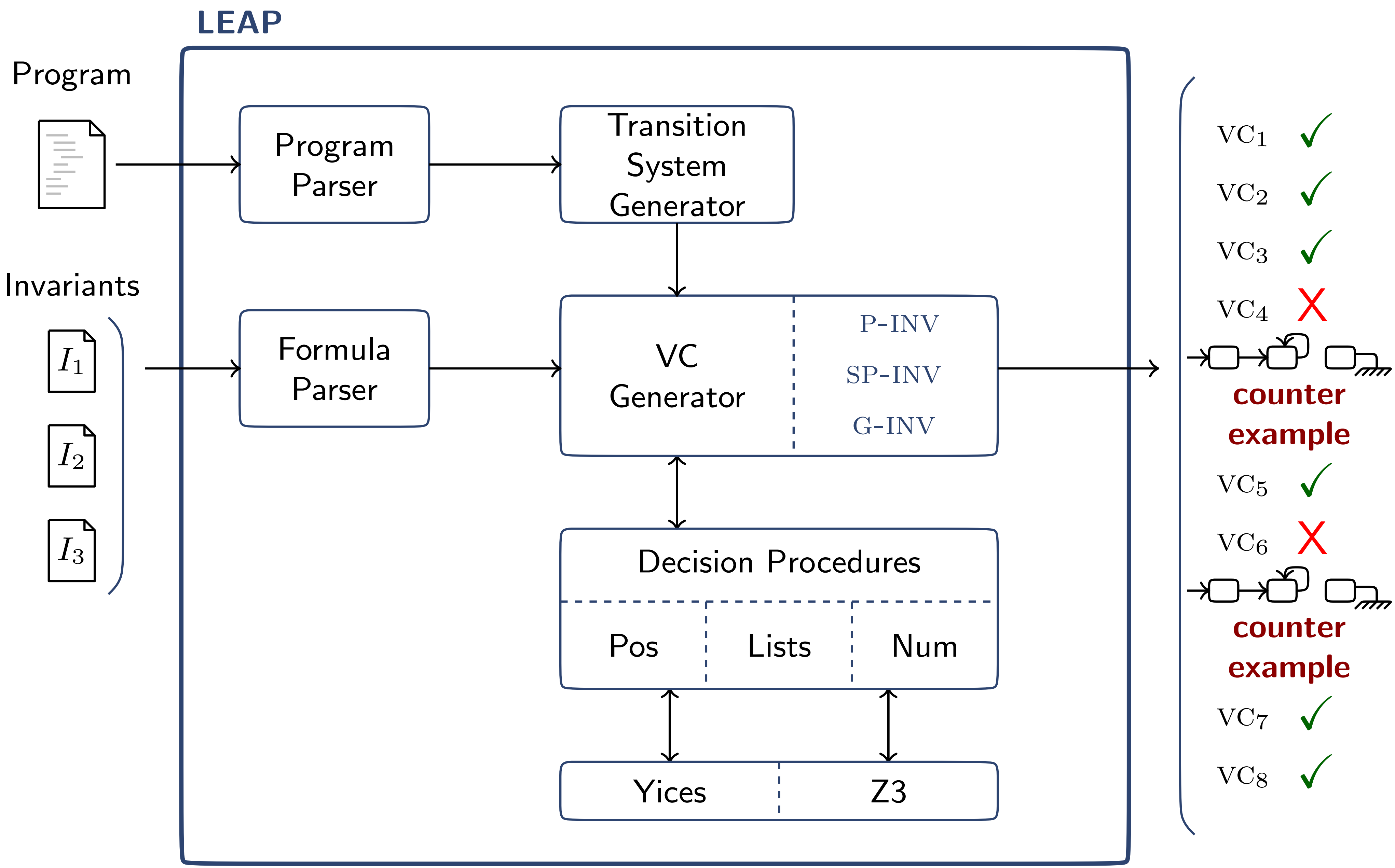
LEAP: A Temporal Deductive Prover



LEAP: A Temporal Deductive Prover



LEAP: A Temporal Deductive Prover

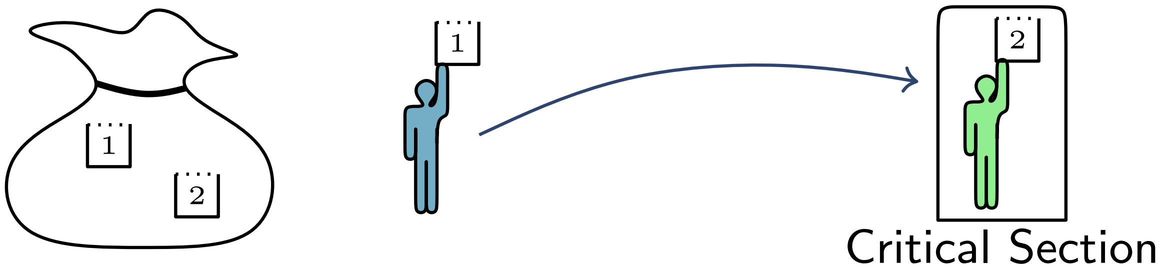


DEMO: Mutex (1st attempt)

- Lets try to prove mutex using P-INV...

DEMO: Mutex (1st attempt)

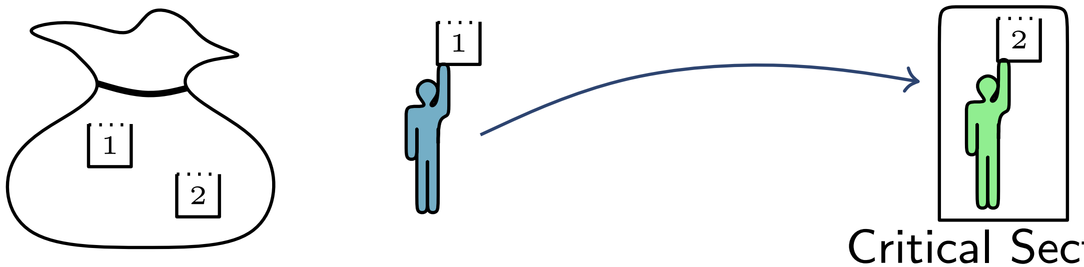
- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



Because mutex does not encode that the
thread in the critical section owns the minimum ticket

DEMO: Mutex (1st attempt)

- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



Because mutex does not encode that the
thread in the critical section owns the minimum ticket

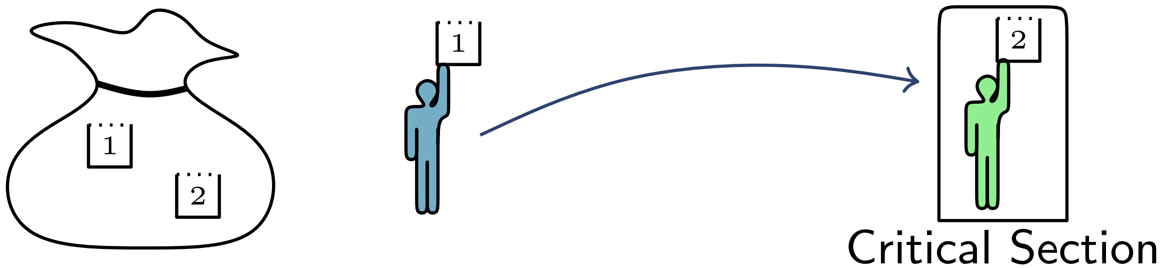
- Extra support is required

$$\text{minticket}(i) \quad \hat{=} \quad \Box\left[critical(i) \rightarrow min(bag) = ticket[i]\right]$$

$$\text{notsame}(i, j) \quad \hat{=} \quad \Box\left[i \neq j \wedge \text{active}(i) \wedge \text{active}(j) \rightarrow ticket(i) \neq ticket(j)\right]$$

DEMO: Mutex (1st attempt)

- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



> Because mutex does not encode that the **thread in the critical section owns the minimum ticket**

- Extra support is required

$$\text{minticket}(i) \;\hat{=}\; \Box\left[critical(i) \rightarrow min(bag) = ticket[i]\right]$$

$$\text{notsame}(i,j) \;\hat{=}\; \Box\left[i \neq j \wedge \text{active}(i) \wedge \text{active}(j) \rightarrow ticket(i) \neq ticket(j)\right]$$

> We now require a **new rule** for **invariant support**

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{lllcll}
(\ \mathsf{S}\ ) & & \Box\psi & & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \text{, forall } i \in \overline{v} \\
(\mathsf{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau \text{, fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

strenghtening

$$
\frac{
\begin{array}{llrcll}
(\text{S}) & & \Box\psi & & & \\
(\text{I}) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ forall } i \in \overline{v} \\
(\text{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \notin \overline{v}
\end{array}
}{\Box\varphi}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\frac{
\begin{array}{llrcll}
(\text{S}) & & \Box\psi & & & \text{strenghtening} \\
(\text{I}) & & \Theta & \rightarrow & \varphi & \text{initiation} \\
(\text{SC}) & \psi,\varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{self-consecution; forall } \tau\text{, forall } i \in \overline{v} \\
(\text{OC}) & \psi,\varphi \rhd & \bigwedge_{x\in\overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{other-consecution; forall } \tau\text{, fresh } k \notin \overline{v}
\end{array}
}{\Box\varphi}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{llccll}
(\text{ S }) & & \Box\psi & & & \\
(\text{ I }) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ forall } i \in \overline{v} \\
(\text{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
\hline
& & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

$\psi \rhd (A \rightarrow B)$ whether $\left[\left(\bigwedge_{\sigma \in S} \psi_\sigma \wedge A\right) \rightarrow B\right]$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{llrcll}
(\text{S}) & & \Box\psi & & & \\
(\text{I}) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ forall } i \in \overline{v} \\
(\text{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
\hline
& & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{llrcll}
(\text{ S }) & & \Box\psi & & & \\
(\text{ I }) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ forall } i \in \overline{v} \\
(\text{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

(S) $\Box$minticket $\wedge$ $\Box$notsame

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

(S)		$\Box\psi$				
(I)		Θ	$\rightarrow$	φ		
(SC)	$\psi, \varphi \rhd$	$\tau^{(i)}$	$\rightarrow$	φ'	forall τ, forall $i \in \overline{v}$
(OC)	$\psi, \varphi \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ'	forall τ, fresh $k \notin \overline{v}$

$$\frac{}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

(S)	$\Box$minticket $\wedge$ $\Box$notsame		
(I)	$\Theta(i, j)$	$\rightarrow$	mutex

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$\begin{array}{llcclll}
(\text{S}) & & \Box\psi & & & & \\
(\text{I}) & & \Theta & \rightarrow & \varphi & & \\
(\text{SC}) & \psi,\varphi \triangleright & \tau^{(i)} & \rightarrow & \varphi' & & \text{forall } \tau \text{, forall } i \in \overline{v} \\
(\text{OC}) & \psi,\varphi \triangleright & \bigwedge_{x\in\overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & & \text{forall } \tau \text{, fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & & & &
\end{array}$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

$$\begin{array}{lccc}
(\text{S}) & \Box\text{minticket} \wedge \Box\text{notsame} & & \\
(\text{I}) & \Theta(i,j) & \rightarrow & \text{mutex} \\
(\text{SC}) & \left[\bigwedge_{\sigma\in\{t_1,\ldots,t_5\}\rightharpoonup\{i,j\}} \begin{bmatrix} \text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{bmatrix}_\sigma \wedge \tau^{(i)} \right] & \rightarrow & \text{mutex}' \quad \forall\tau
\end{array}$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{llrcll}
(\text{S}) & & \Box\psi & & & \\
(\text{I}) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi,\varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau\text{, forall } i \in \overline{v} \\
(\text{OC}) & \psi,\varphi \rhd & \bigwedge_{x\in\overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau\text{, fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i,j)

$$
\begin{array}{lrcll}
(\text{S}) & \Box\text{minticket} \wedge \Box\text{notsame} & & & \\
(\text{I}) & \Theta(i,j) & \rightarrow & \text{mutex} & \\
(\text{SC}) & \left[\bigwedge_{\sigma\in\{t_1,\ldots,t_5\}\rightharpoonup\{i,j\}} \begin{bmatrix}\text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5)\end{bmatrix}_\sigma \wedge \tau^{(i)}\right] & \rightarrow & \text{mutex}' & \forall\tau \\
 & \left[\bigwedge_{\sigma\in\{t_1,\ldots,t_5\}\rightharpoonup\{i,j\}} \begin{bmatrix}\text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5)\end{bmatrix}_\sigma \wedge \tau^{(j)}\right] & \rightarrow & \text{mutex}' & \forall\tau
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(\overline{v})$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{llrcll}
(\text{S}) & & \Box\psi & & & \\
(\text{I}) & & \Theta & \rightarrow & \varphi & \\
(\text{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ forall } i \in \overline{v} \\
(\text{OC}) & \psi, \varphi \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

$$
\begin{array}{lrcll}
(\text{S}) & \Box\text{minticket} \wedge \Box\text{notsame} & & & \\
(\text{I}) & \Theta(i,j) & \rightarrow & \text{mutex} & \\
(\text{SC}) & \left[\bigwedge_{\sigma \in \{t_1,\ldots,t_5\} \rightharpoonup \{i,j\}} \begin{bmatrix} \text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{bmatrix}_\sigma \wedge \tau^{(i)} \right] & \rightarrow & \text{mutex}' & \forall\tau \\
 & \left[\bigwedge_{\sigma \in \{t_1,\ldots,t_5\} \rightharpoonup \{i,j\}} \begin{bmatrix} \text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{bmatrix}_\sigma \wedge \tau^{(j)} \right] & \rightarrow & \text{mutex}' & \forall\tau \\
(\text{OC}) & \left[\bigwedge_{\sigma \in \{t_1,\ldots t_5\} \rightharpoonup \{i,j,k\}} \begin{bmatrix} \text{minticket}(t_1) \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{bmatrix}_\sigma \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} \right] & \rightarrow & \text{mutex}' & \forall\tau
\end{array}
$$

DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

Now we can **prove** mutex
using minticket and notsame as **support**

DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

> Now we can **prove** mutex
> using minticket and notsame as **support**

- How does the **full proof** look?

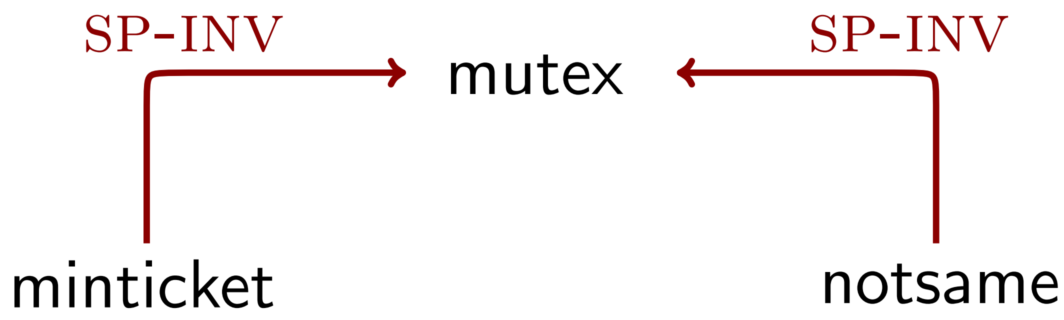
mutex

DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

> Now we can **prove** mutex
> using minticket and notsame as **support**

- How does the **full proof** look?

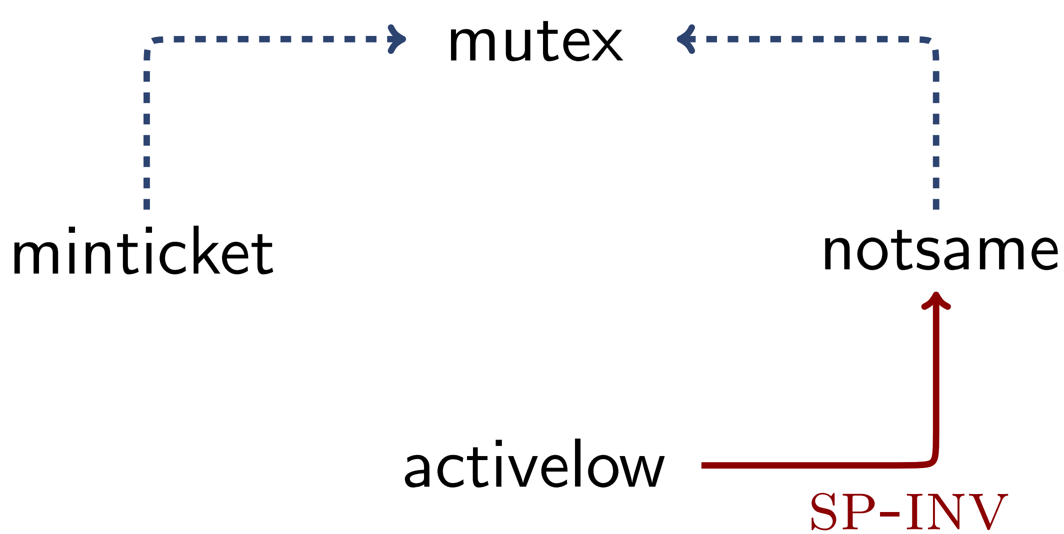


DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

> Now we can **prove** mutex
> using minticket and notsame as **support**

- How does the **full proof** look?

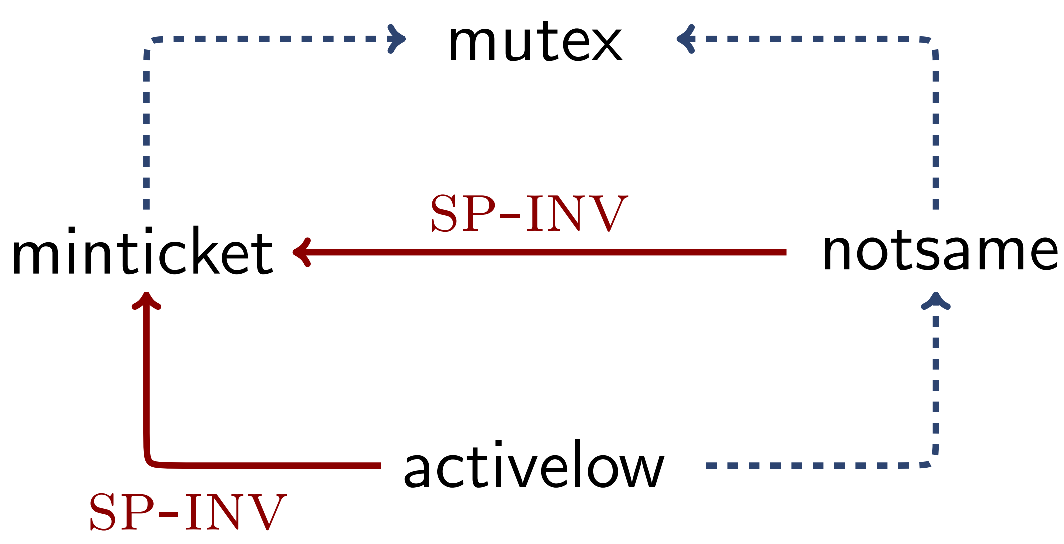


DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

> Now we can **prove** mutex
> using minticket and notsame as **support**

- How does the **full proof** look?

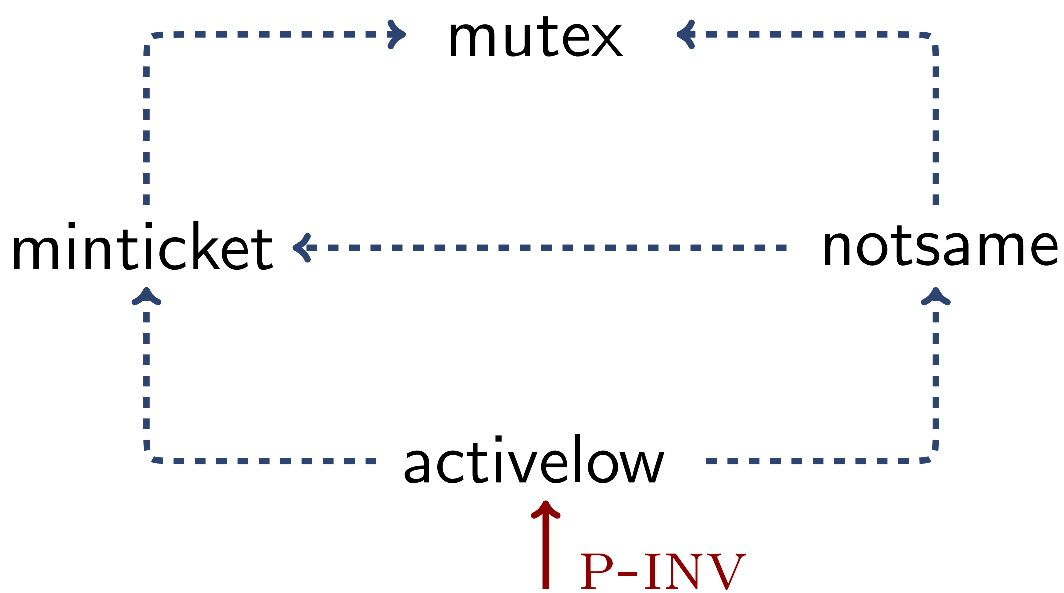


DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

> Now we can **prove** mutex
> using minticket and notsame as **support**

- How does the **full proof** look?

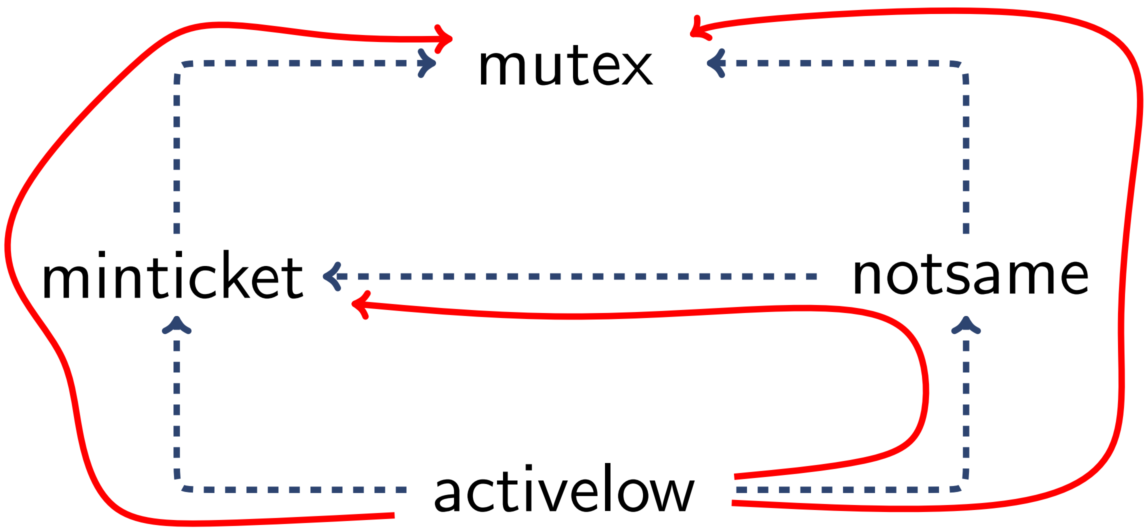


DEMO: Mutex (2nd attempt)

- Lets try to prove mutex using SP-INV with support...

Now we can **prove** mutex
using minticket and notsame as **support**

- How does the **full proof** look?



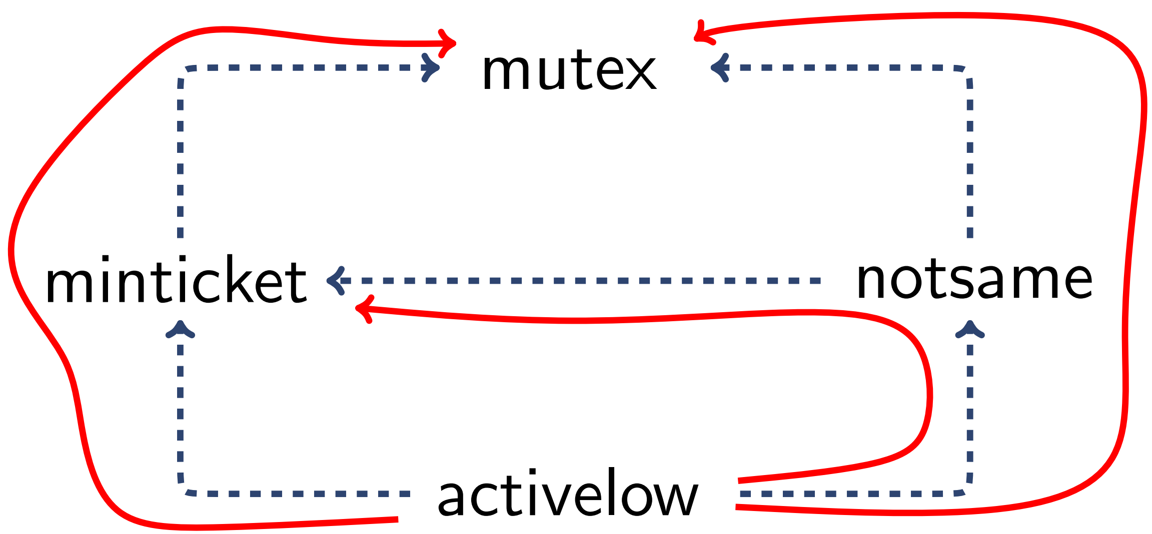
In this case, the proof is a **DAG**

DEMO: Mutex (2nd attempt)

- **LEAP format**

```
-> activeLow

-> notSame [3:SC:activeLow]

-> minTicket [3:activeLow; 6:OC:activeLow,notSame]

-> mutex [4:SC:notSame, minTicket]
```

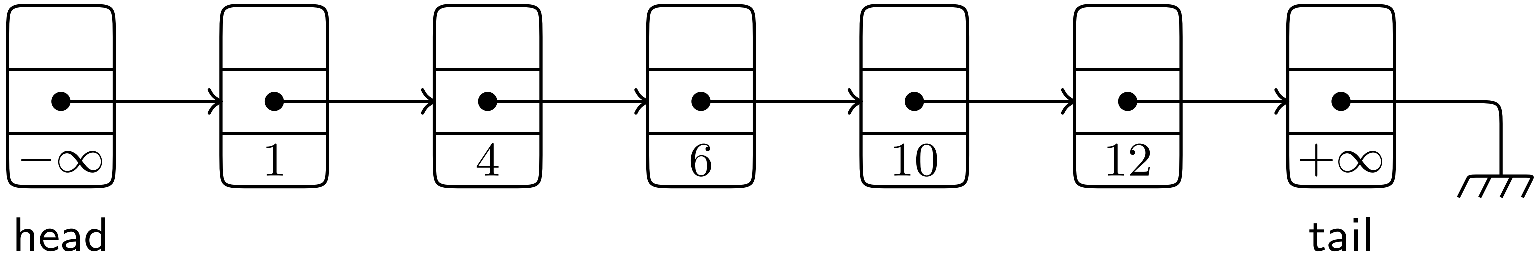
- How does the **full proof** look?



In this case, the proof is a **DAG**

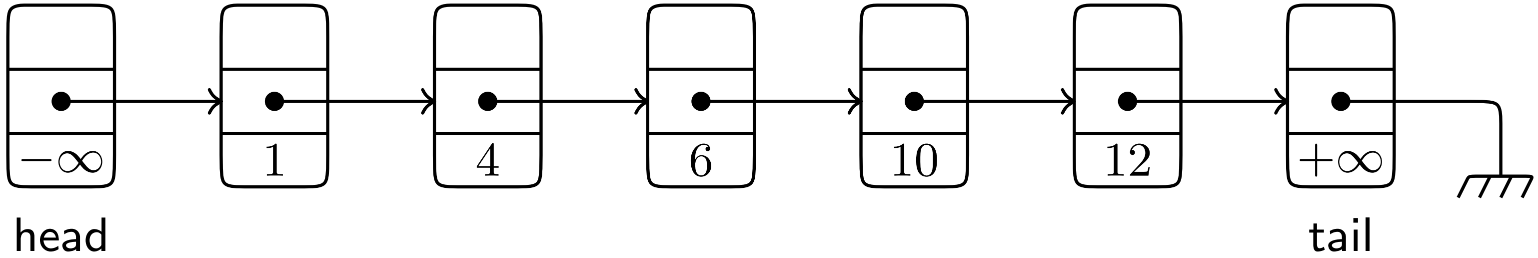
Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets



Case Study 2: Lock-Coupling Lists

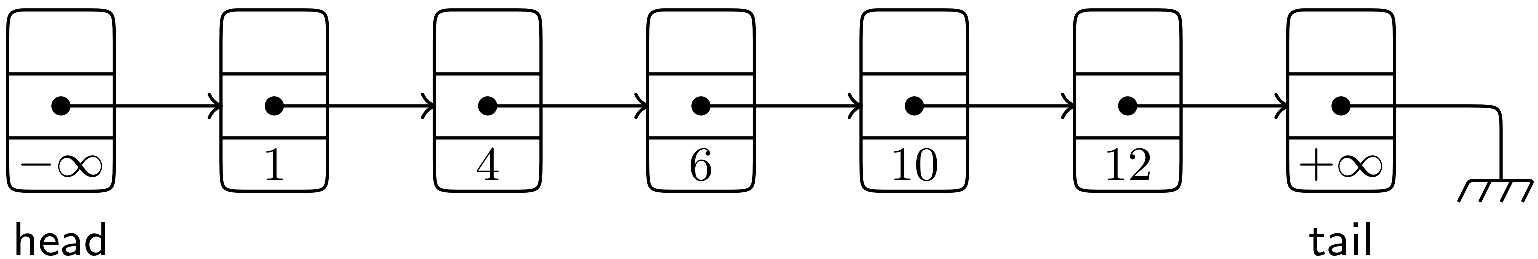
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

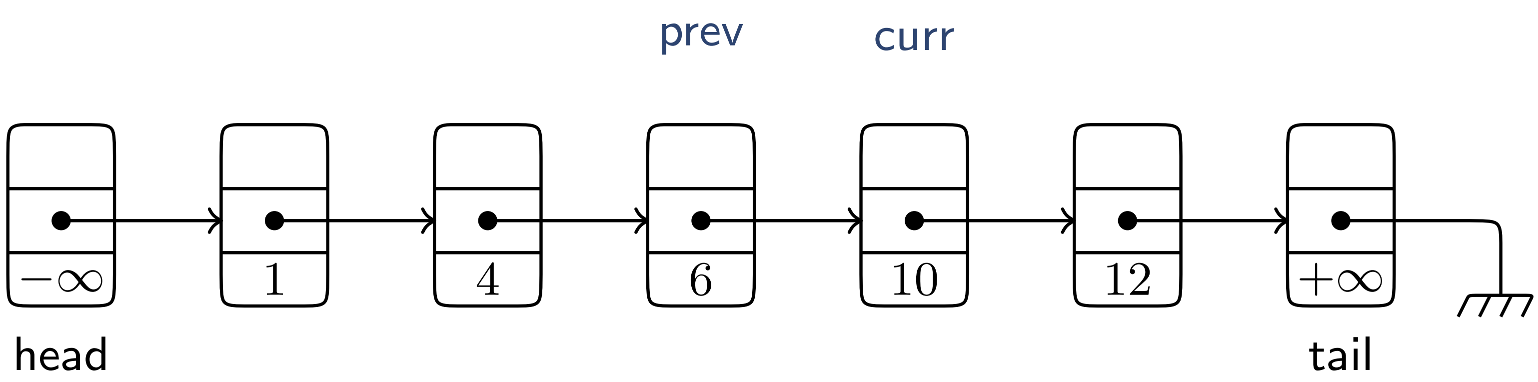
SEARCH(9)



Case Study 2: Lock-Coupling Lists

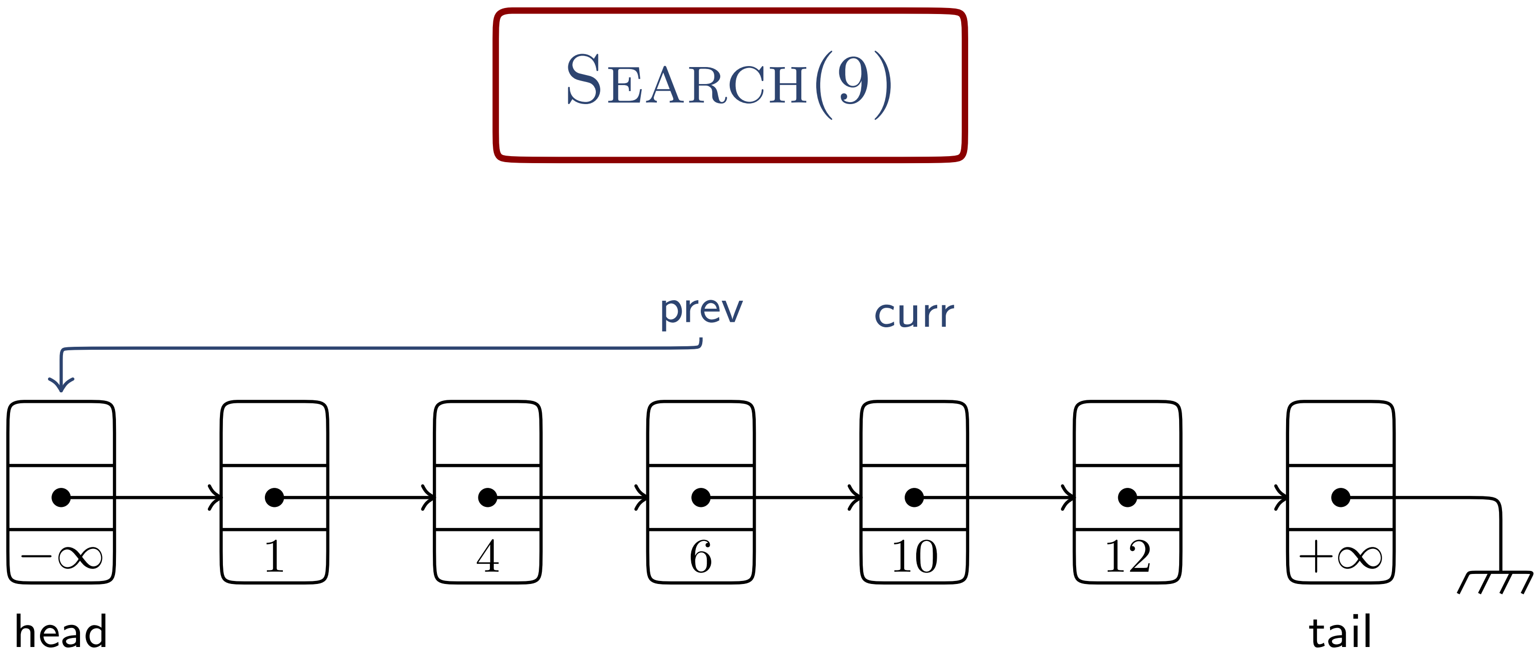
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



Case Study 2: Lock-Coupling Lists

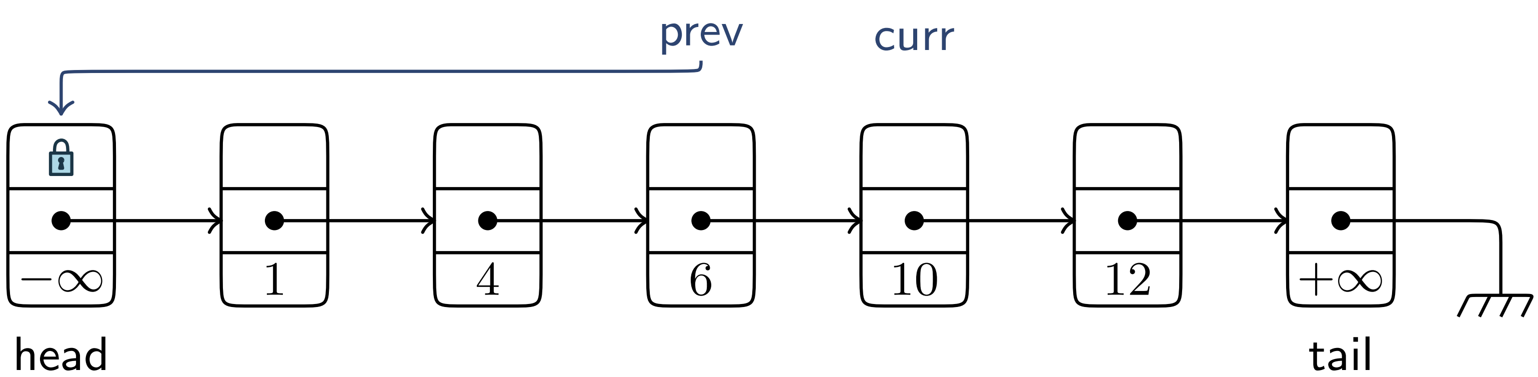
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE



Case Study 2: Lock-Coupling Lists

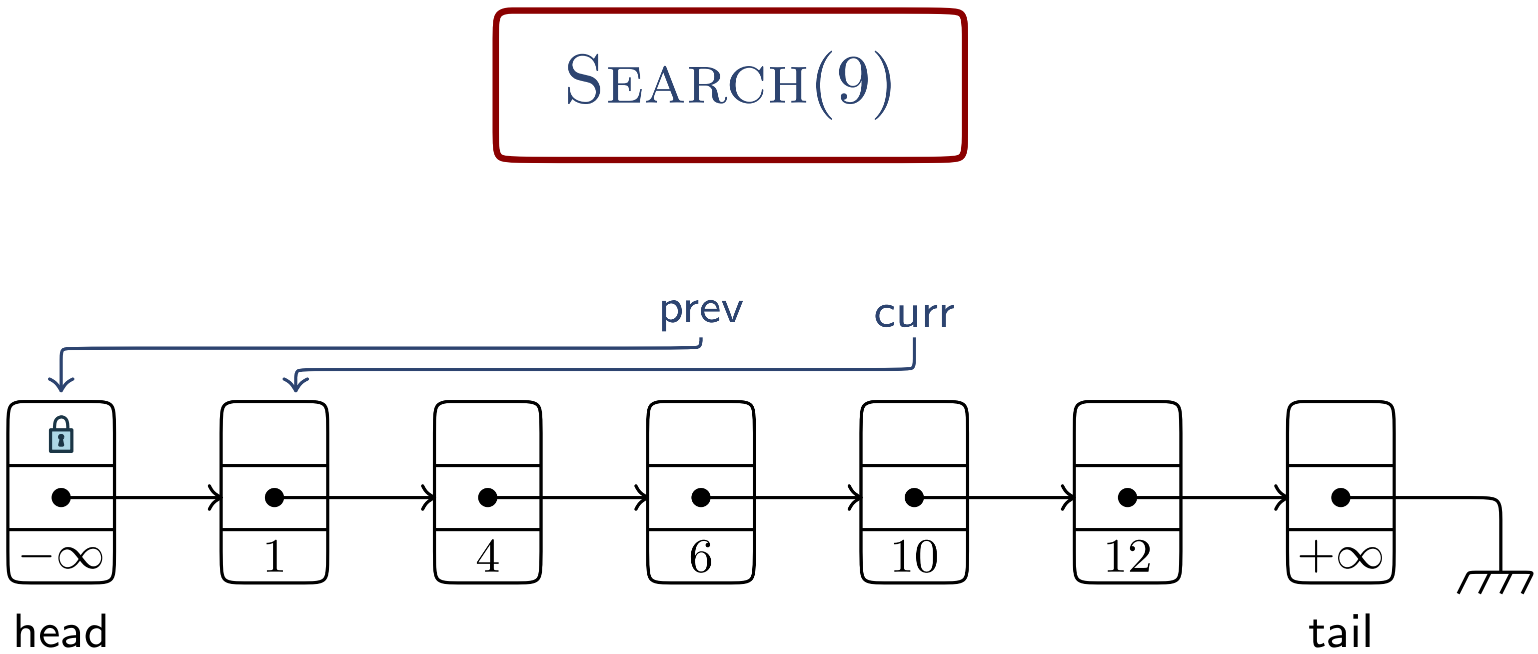
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



Case Study 2: Lock-Coupling Lists

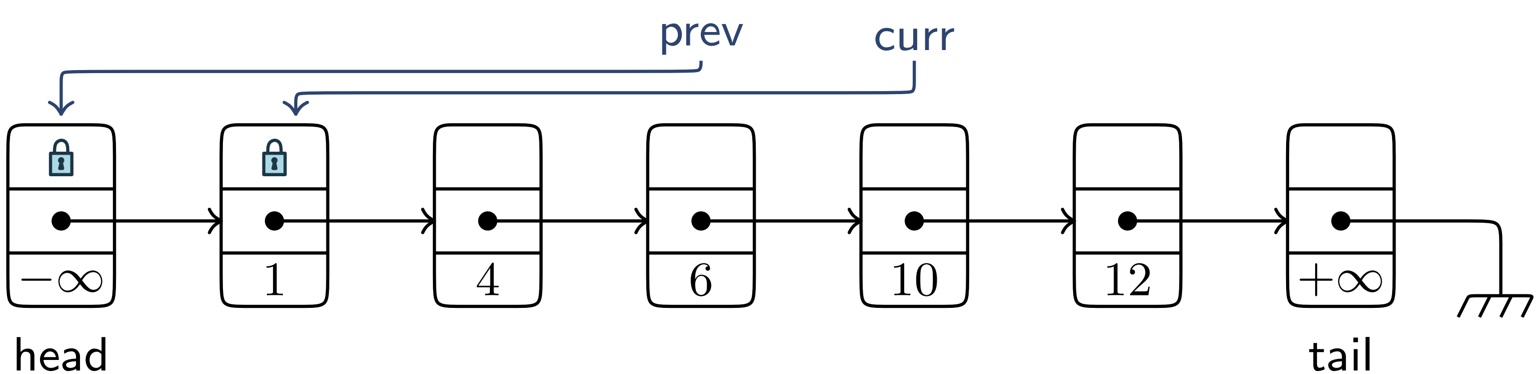
- A concurrent implementation of sets
- Operations such as Search, Insert and Remove



Case Study 2: Lock-Coupling Lists

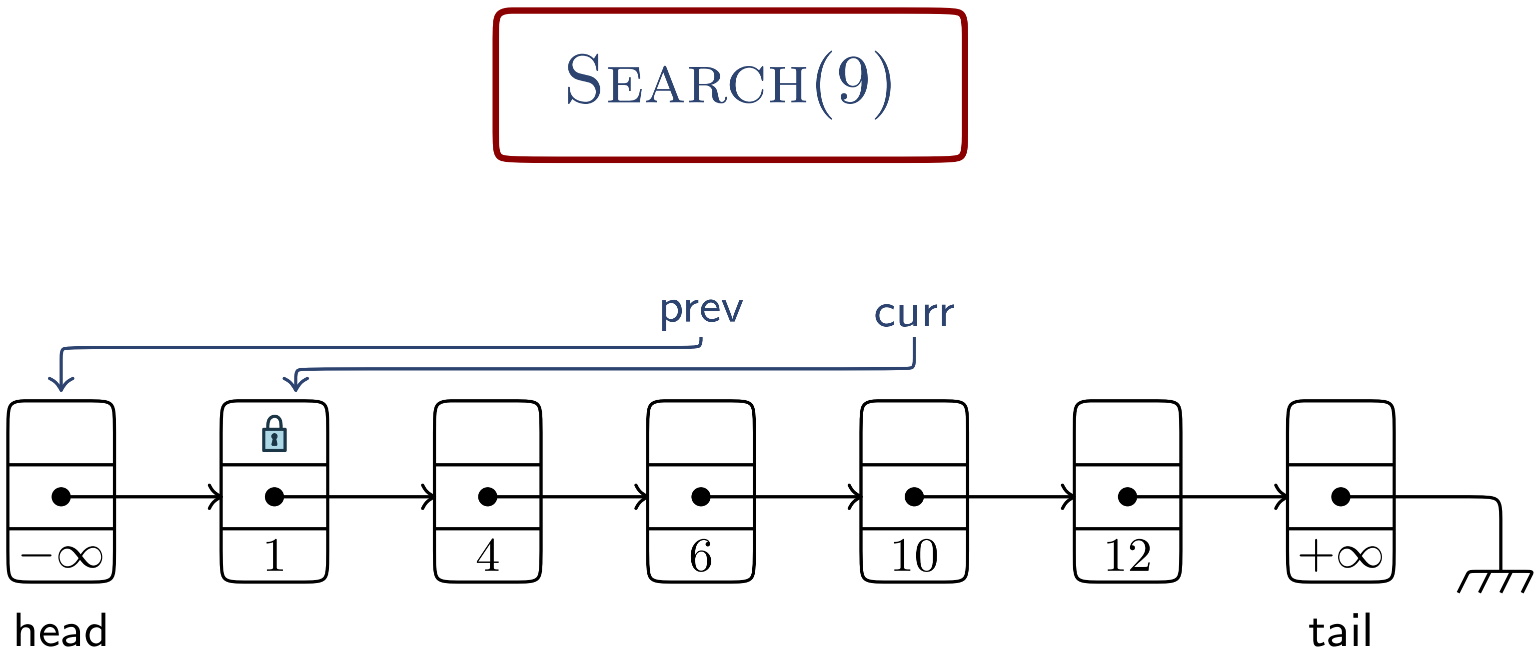
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



Case Study 2: Lock-Coupling Lists

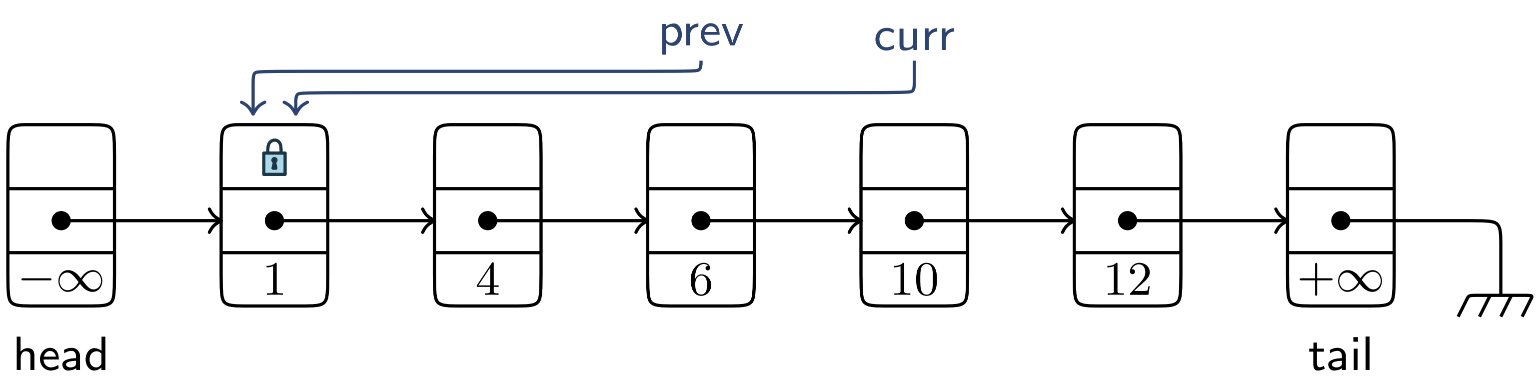
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE



Case Study 2: Lock-Coupling Lists

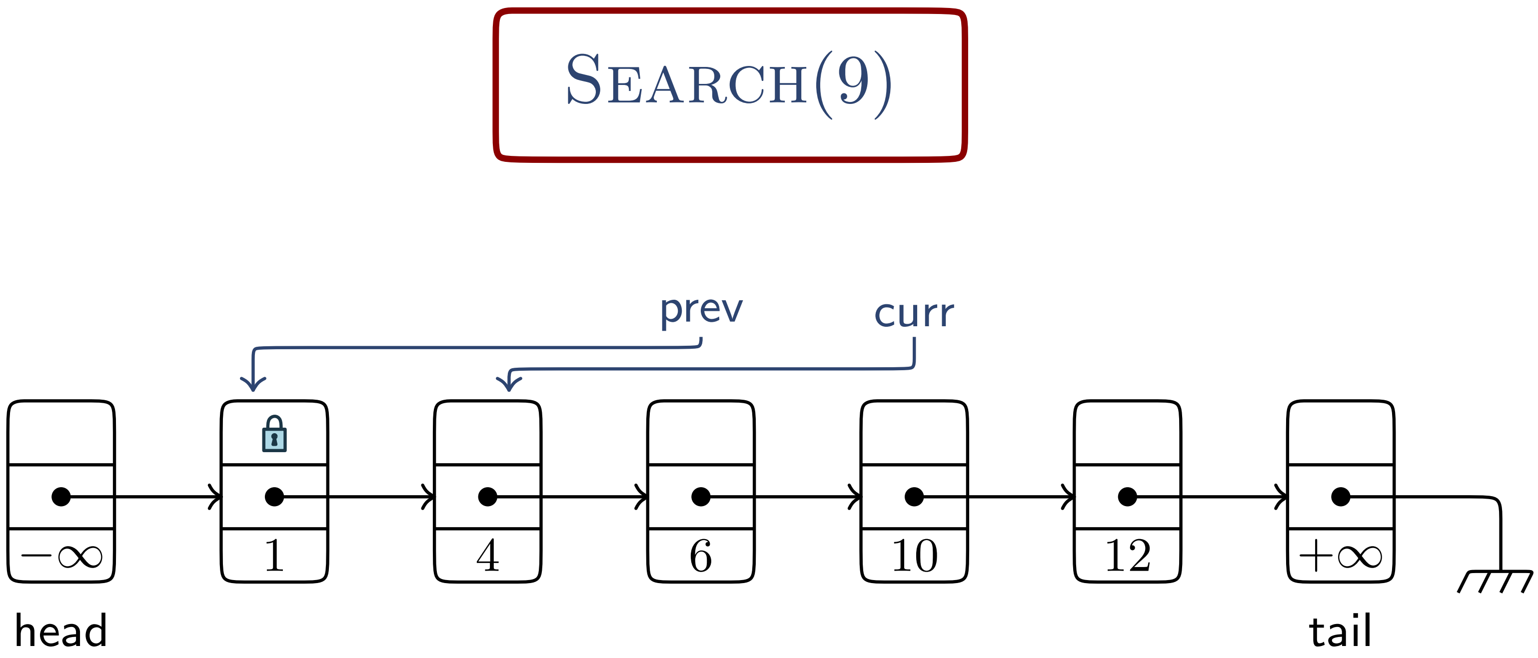
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



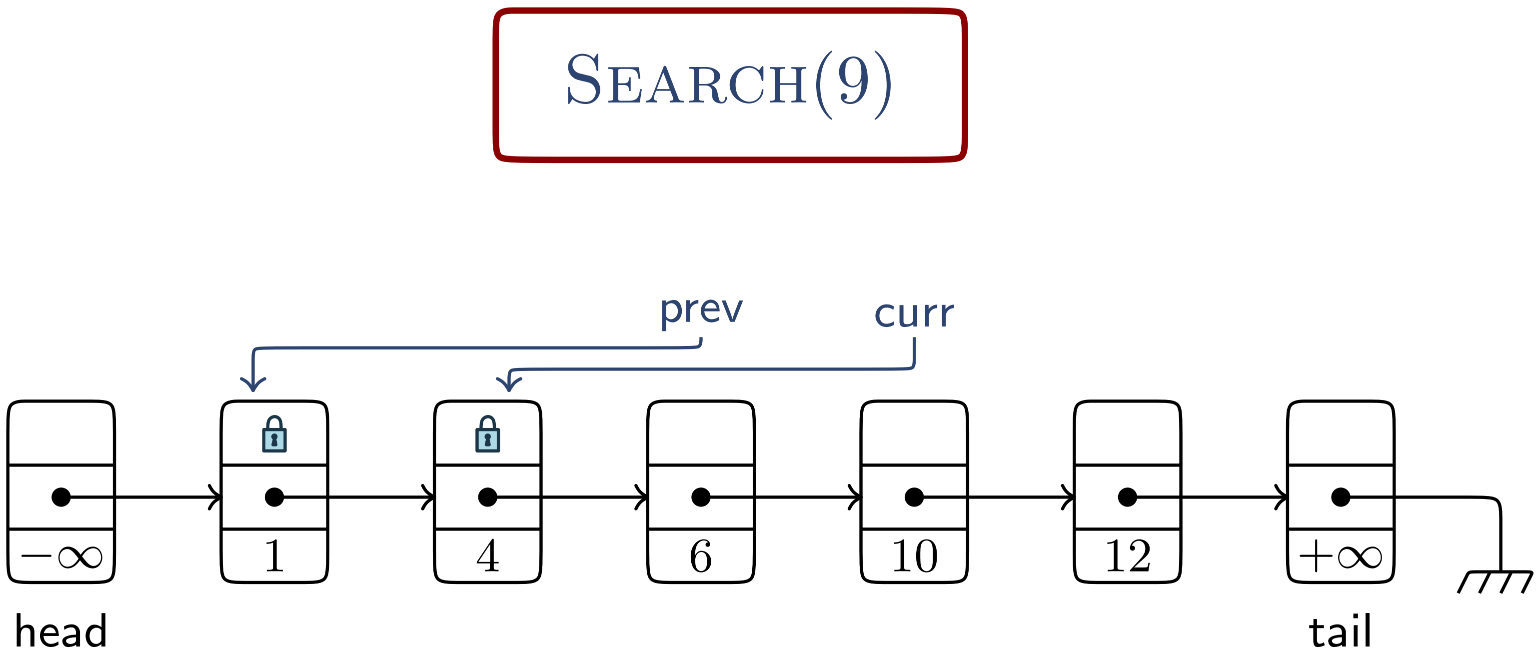
Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as Search, Insert and Remove



Case Study 2: Lock-Coupling Lists

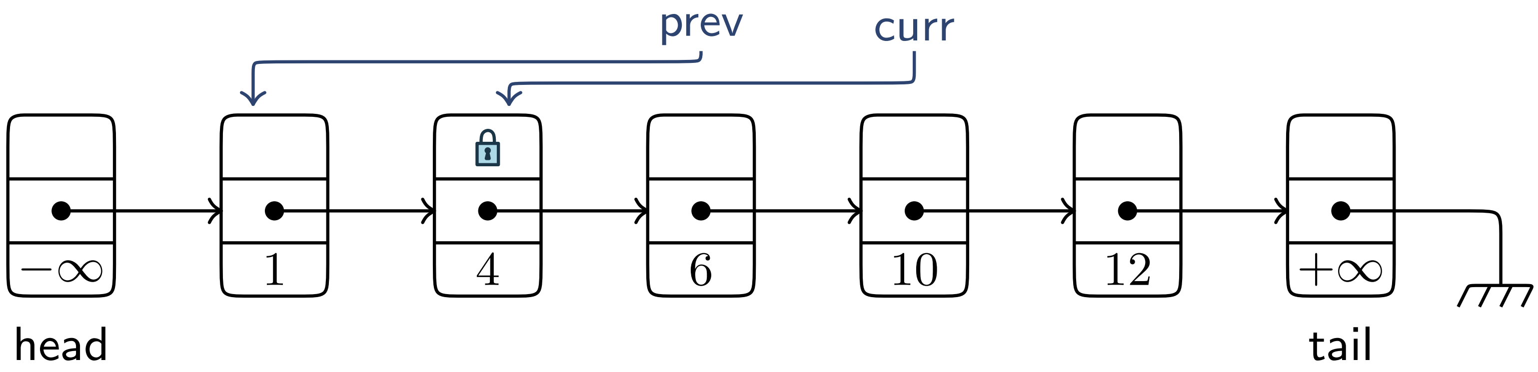
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE



Case Study 2: Lock-Coupling Lists

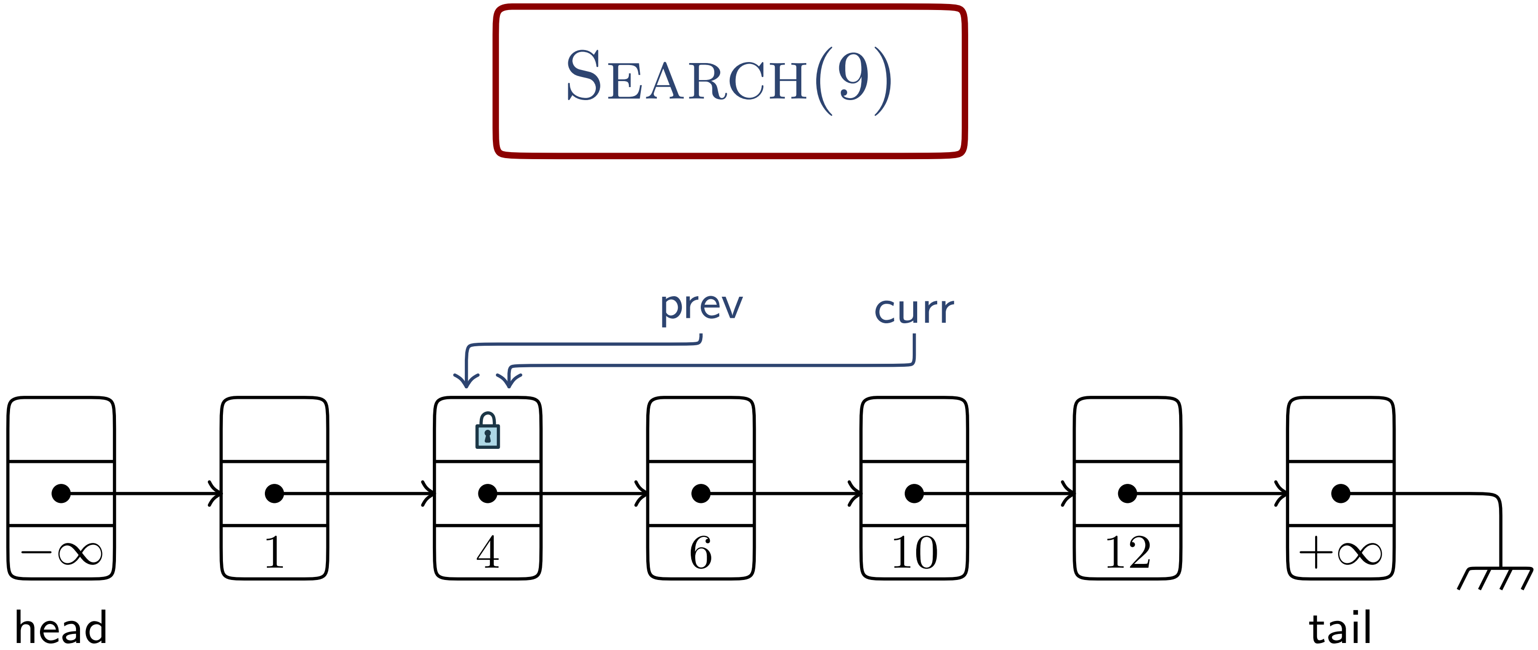
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



Case Study 2: Lock-Coupling Lists

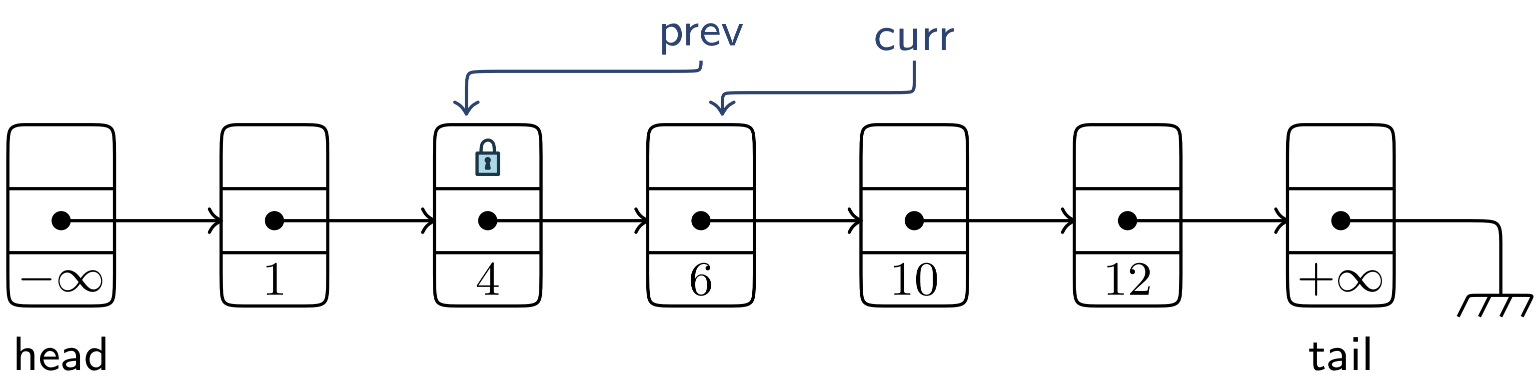
- A concurrent implementation of sets
- Operations such as Search, Insert and Remove



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as Search, Insert and Remove

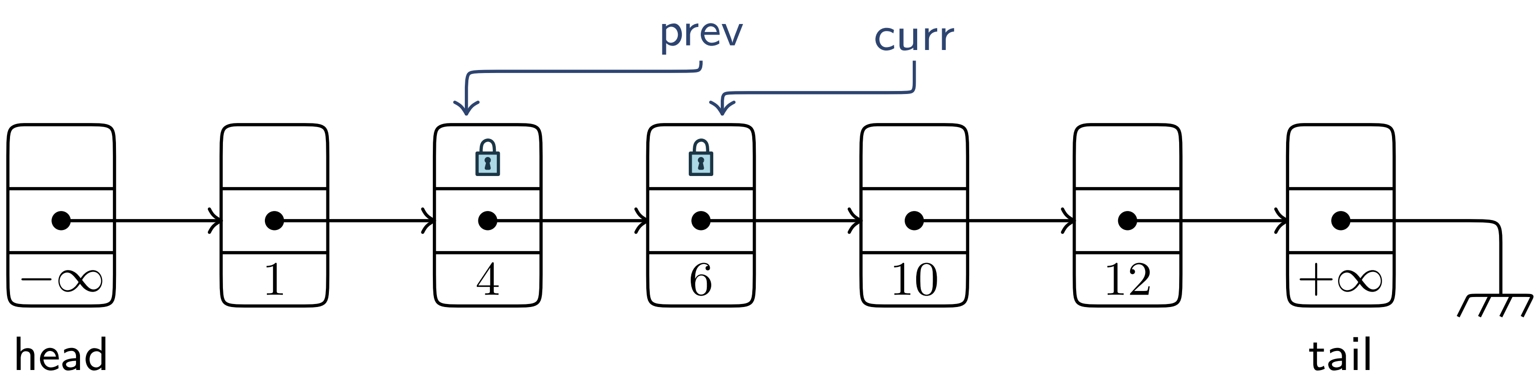
Search(9)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as Search, Insert and Remove

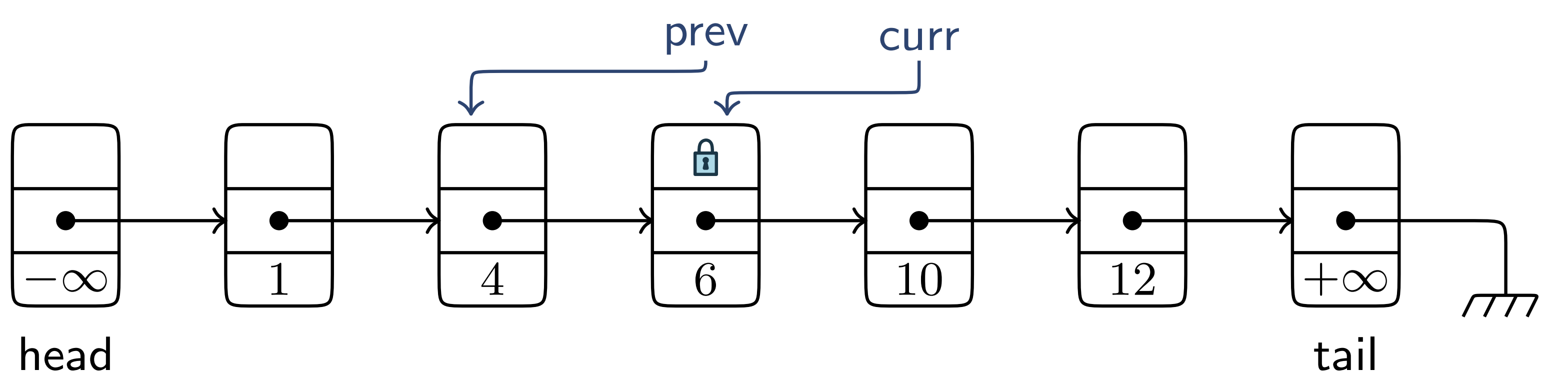
Search(9)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as Search, Insert and Remove

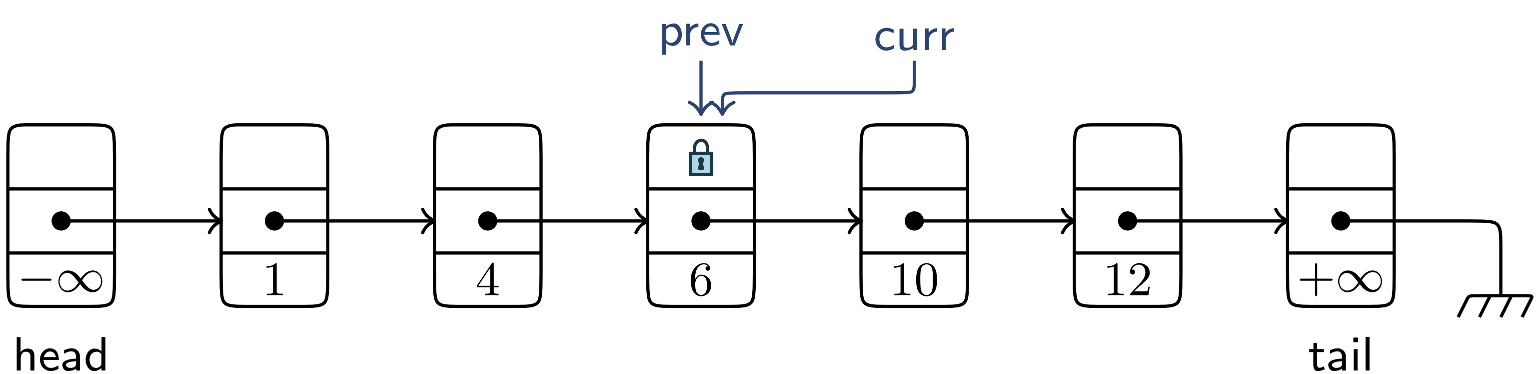
Search(9)



Case Study 2: Lock-Coupling Lists

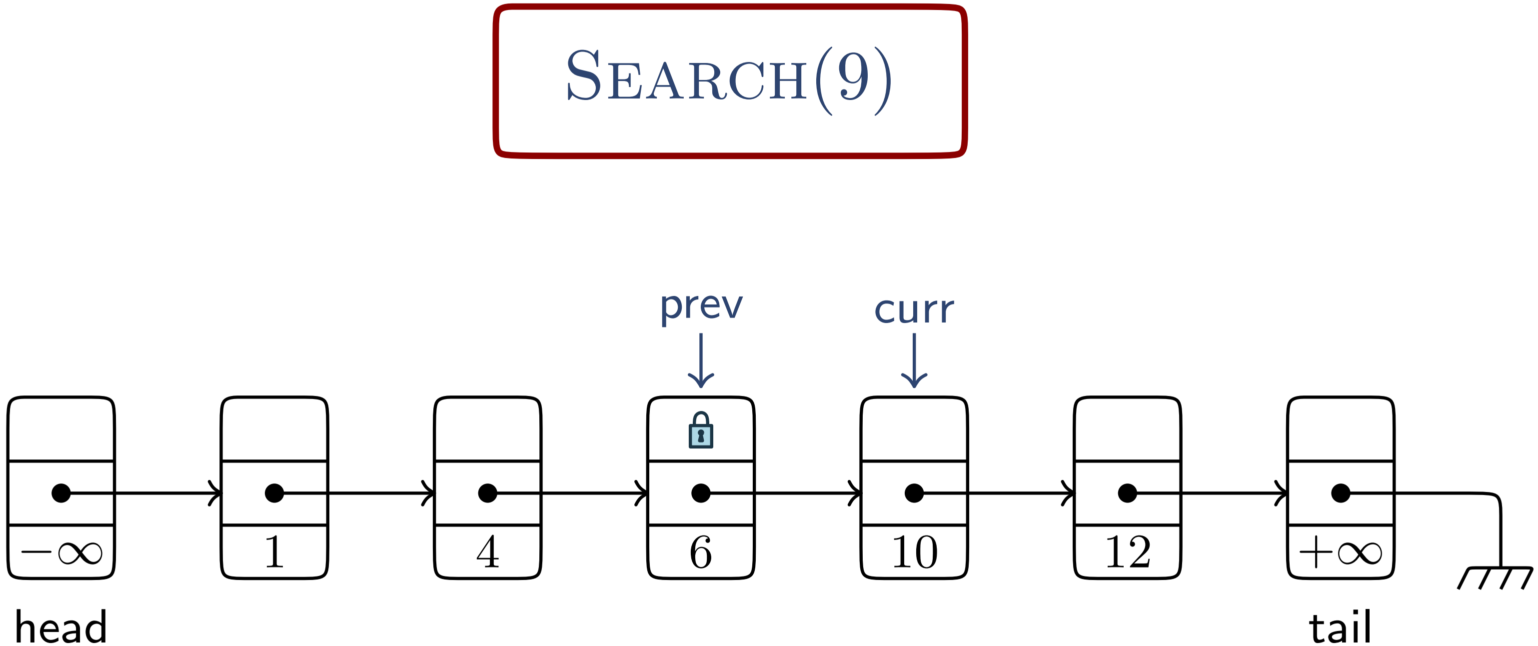
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

SEARCH(9)



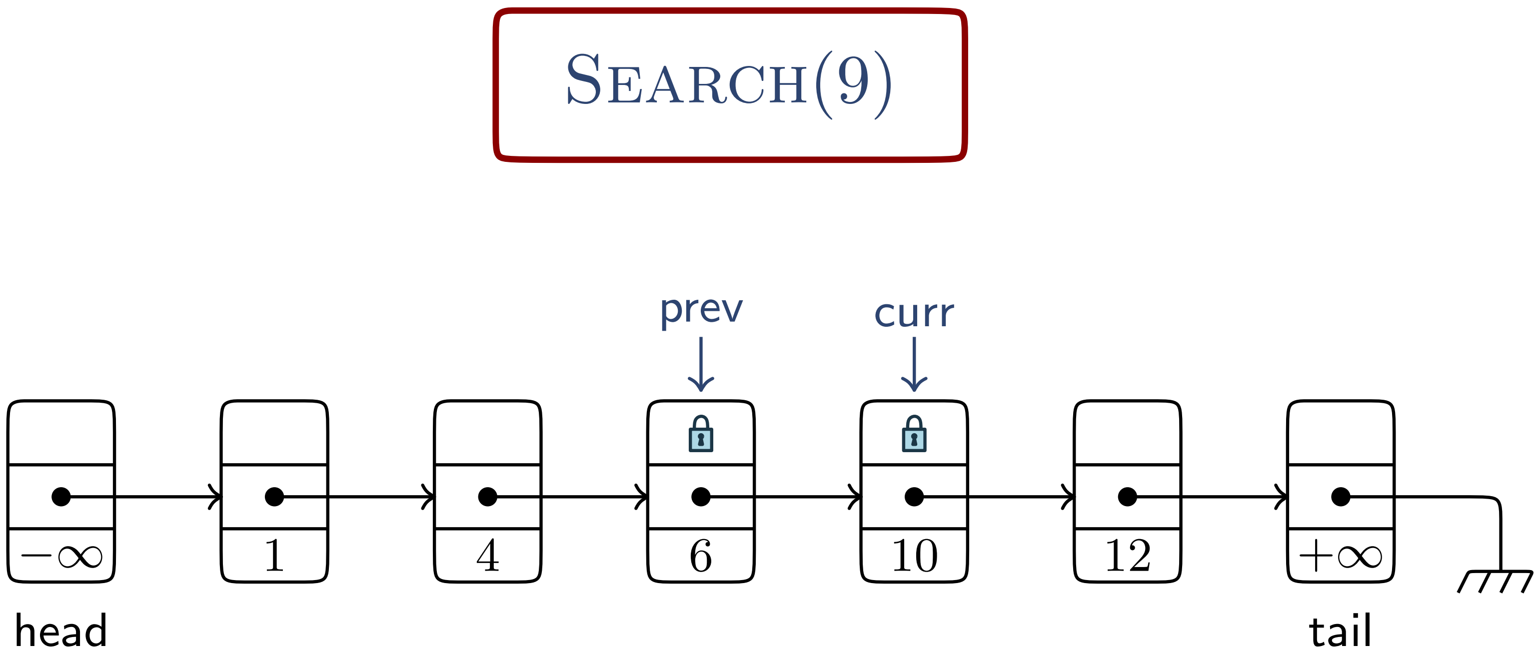
Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as Search, Insert and Remove



Case Study 2: Lock-Coupling Lists

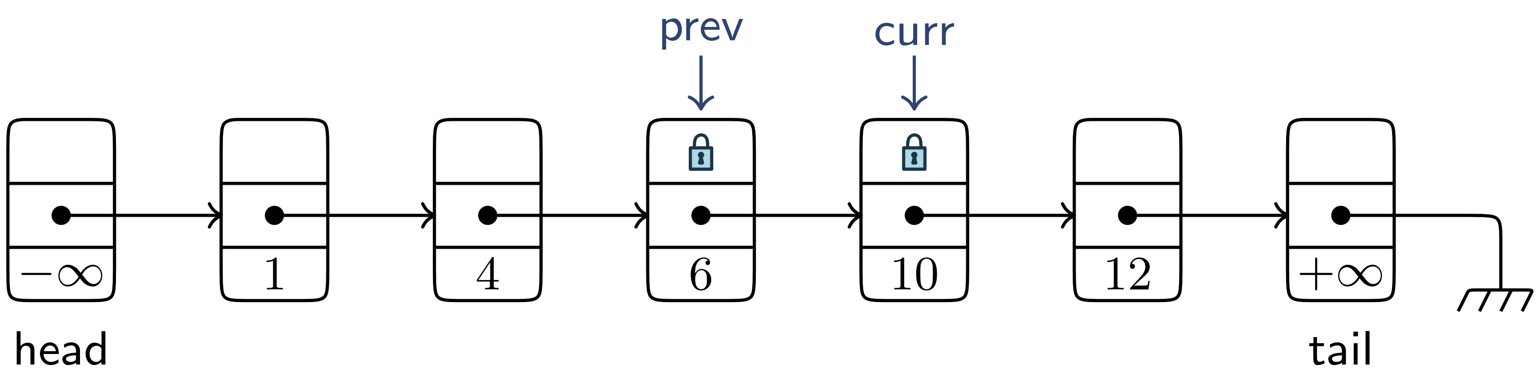
- A concurrent implementation of sets
- Operations such as Search, Insert and Remove



Case Study 2: Lock-Coupling Lists

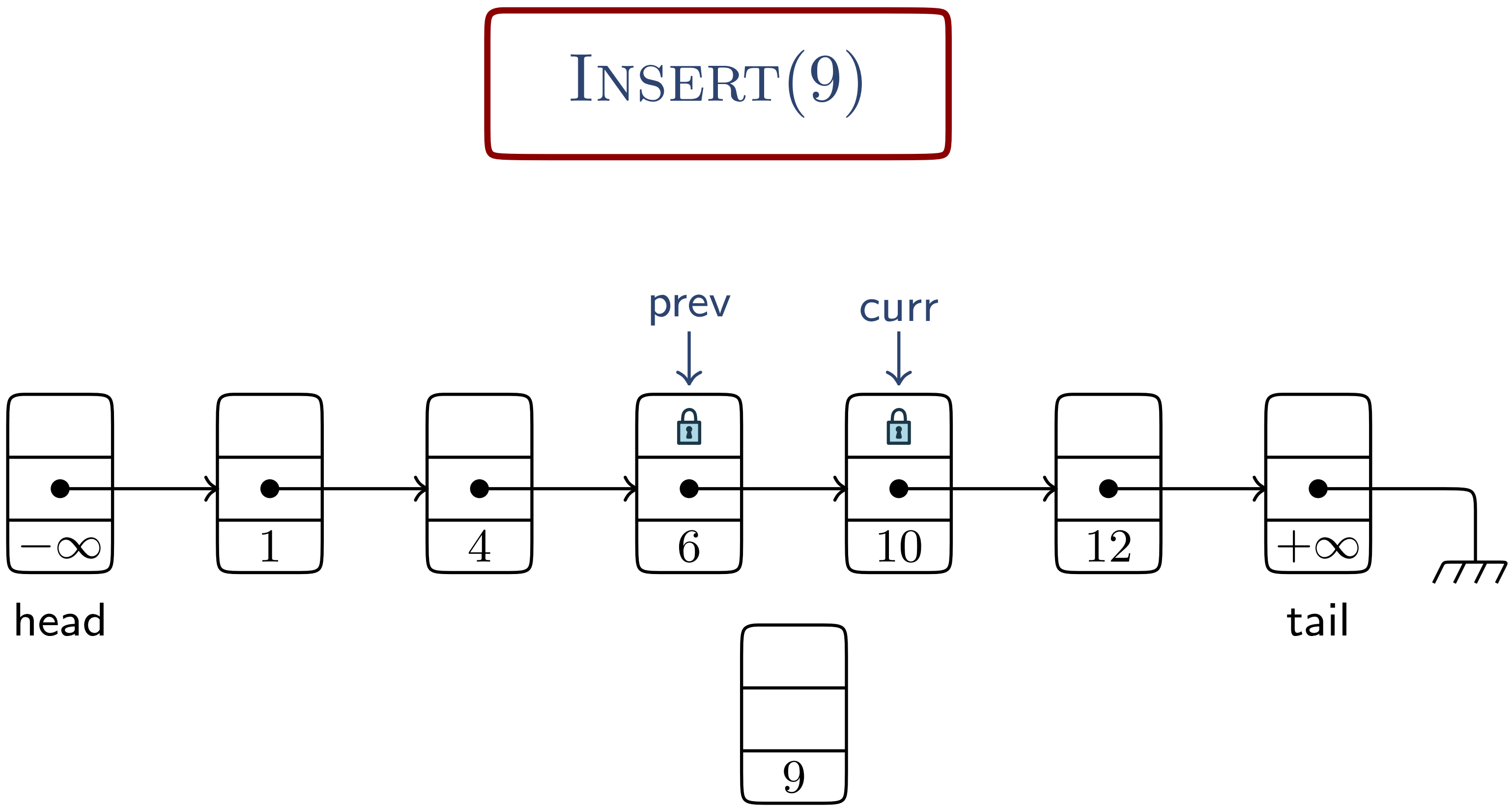
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

INSERT(9)



Case Study 2: Lock-Coupling Lists

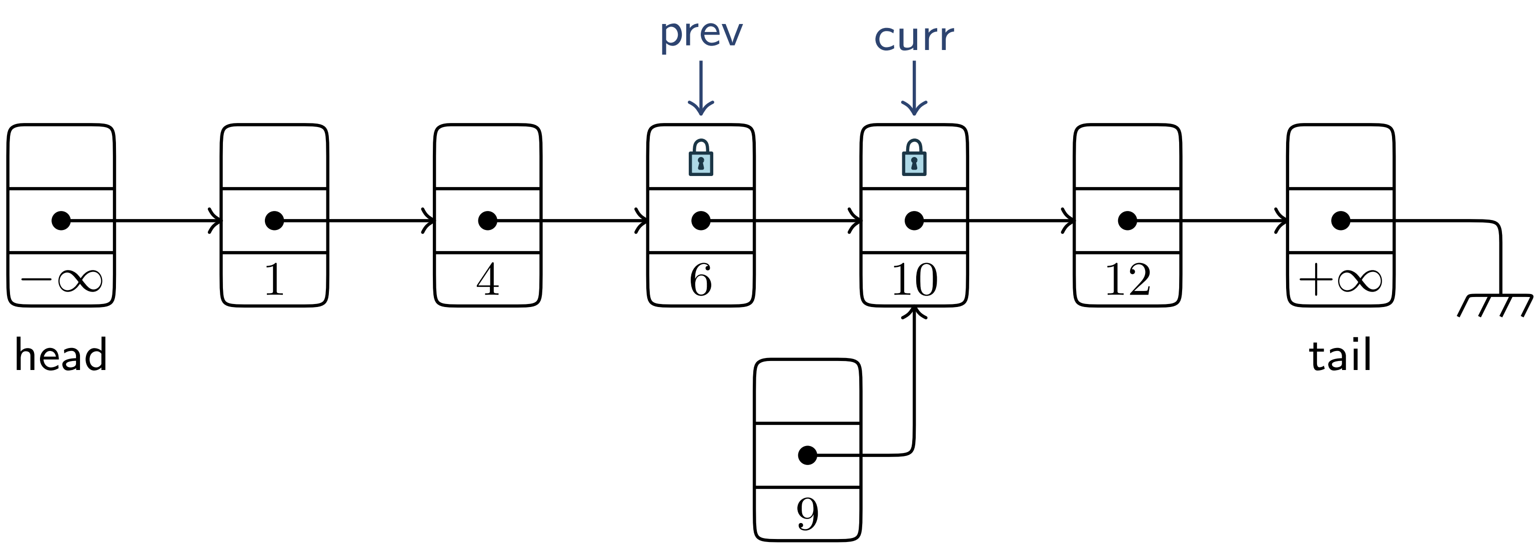
- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

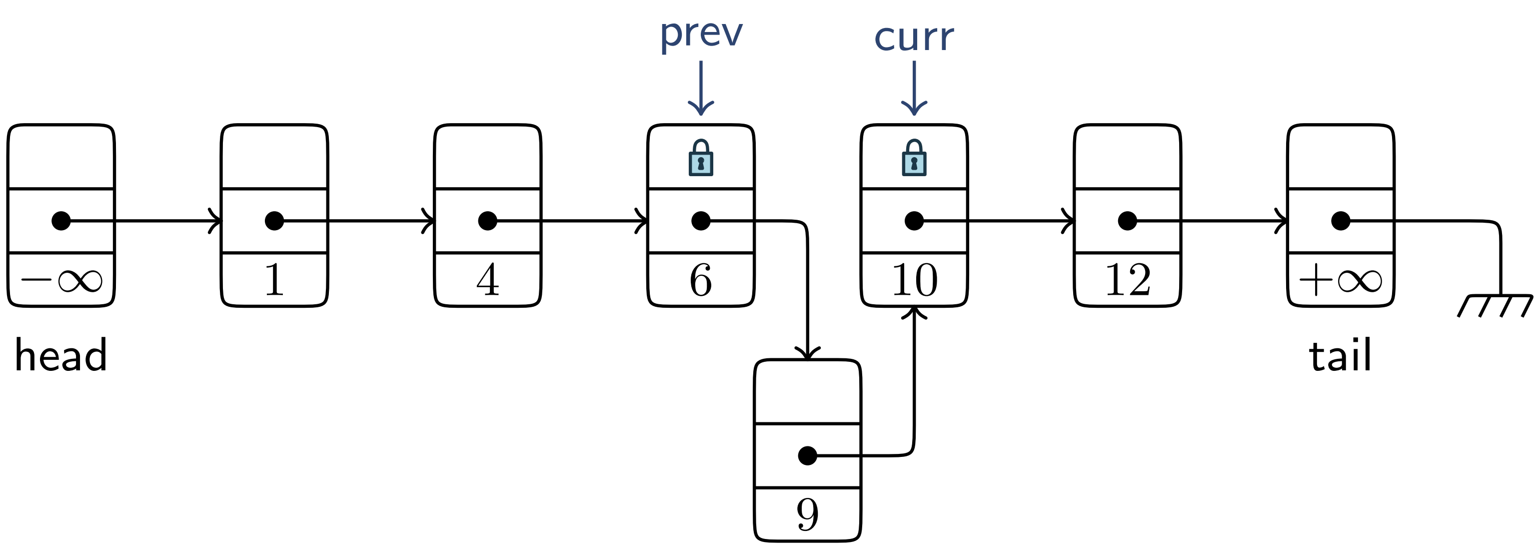
INSERT(9)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

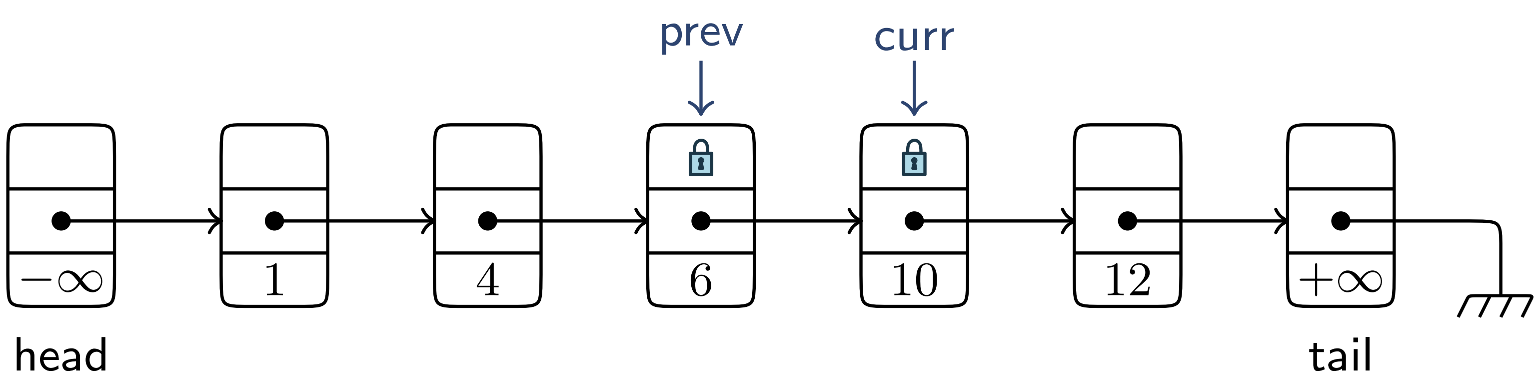
INSERT(9)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

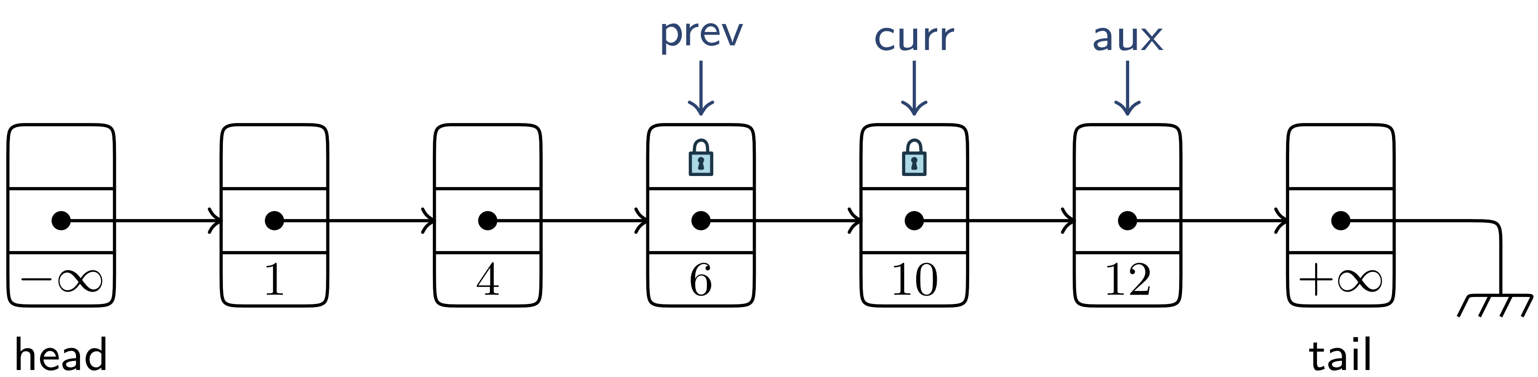
REMOVE(10)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

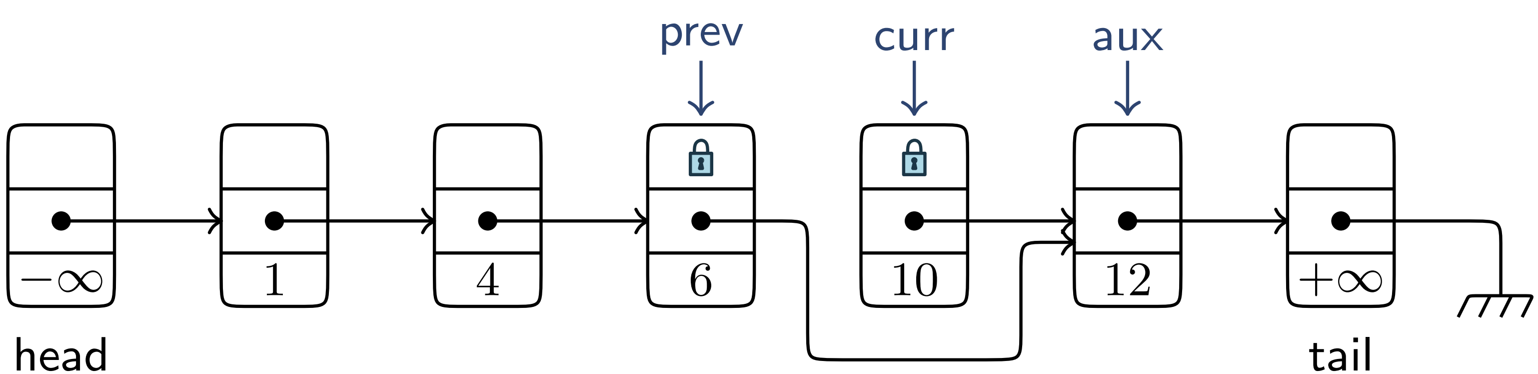
REMOVE(10)



Case Study 2: Lock-Coupling Lists

- A concurrent implementation of sets
- Operations such as SEARCH, INSERT and REMOVE

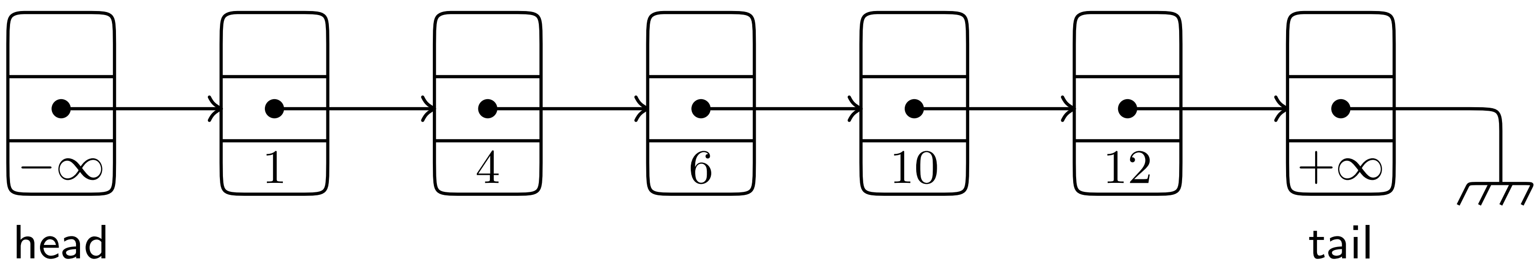
REMOVE(10)



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

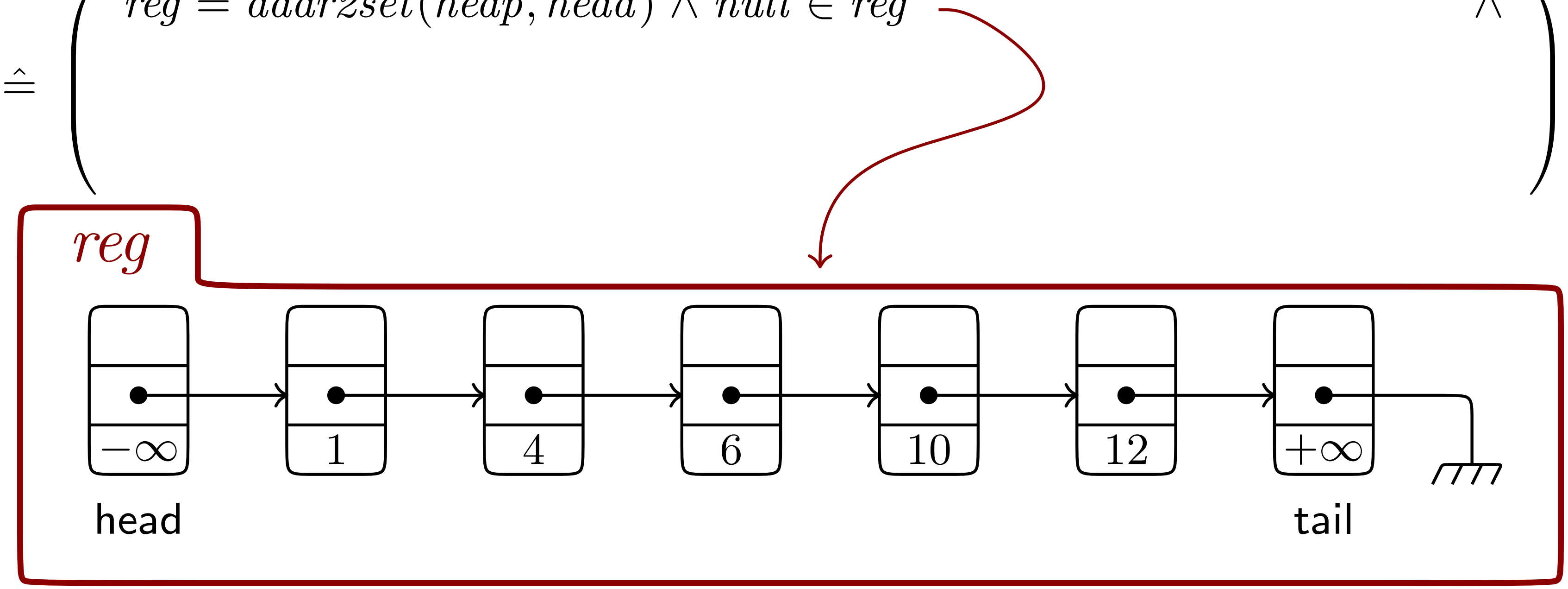
$$\text{list} \triangleq \left(\qquad \right)$$



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

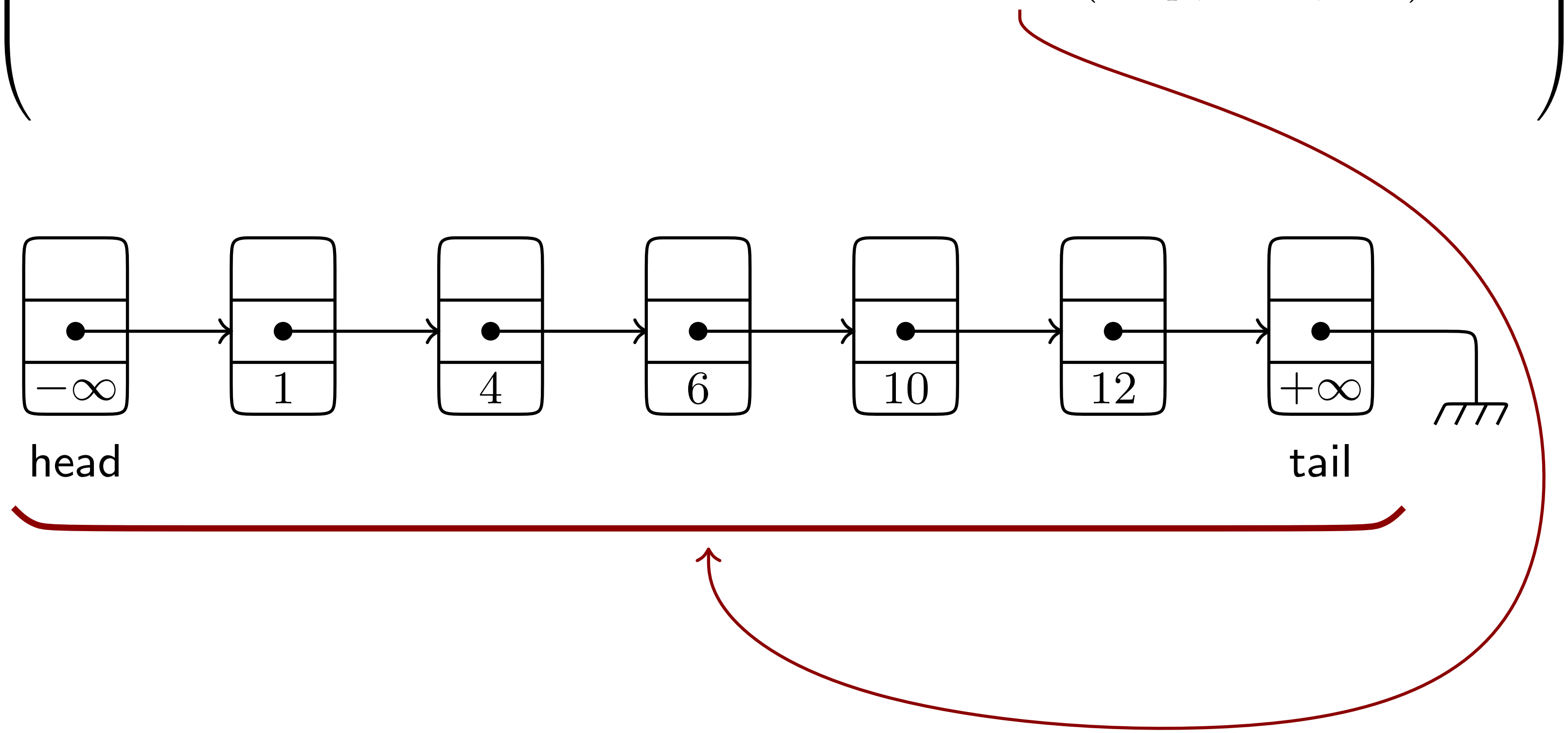
$$\text{list} \triangleq \left(\begin{array}{l} reg = addr2set(heap, head) \wedge null \in reg \qquad \wedge \\ \\ \end{array} \right)$$



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

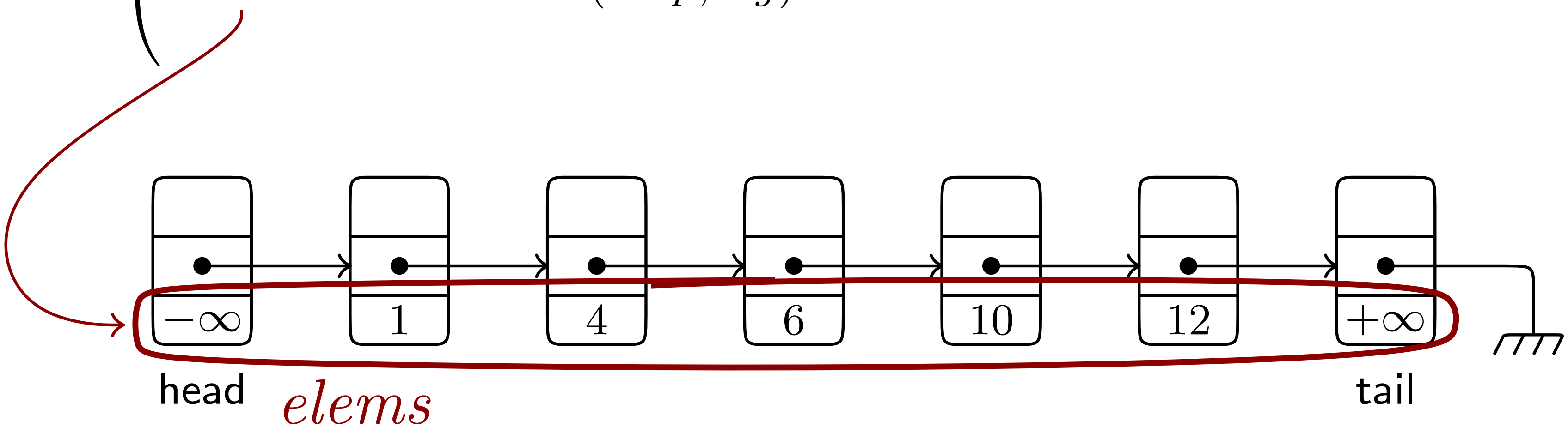
$$\text{list} \mathrel{\hat{=}} \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\rightarrow}data = -\infty \wedge tail{\rightarrow}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ & \end{array} \right)$$



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

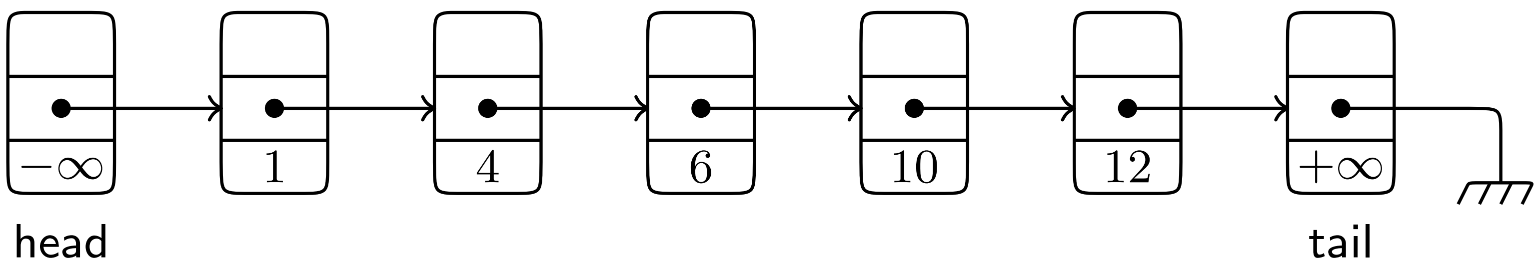
$$\text{list} \mathrel{\hat{=}} \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\rightarrow}data = -\infty \wedge tail{\rightarrow}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \end{array} \right)$$



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

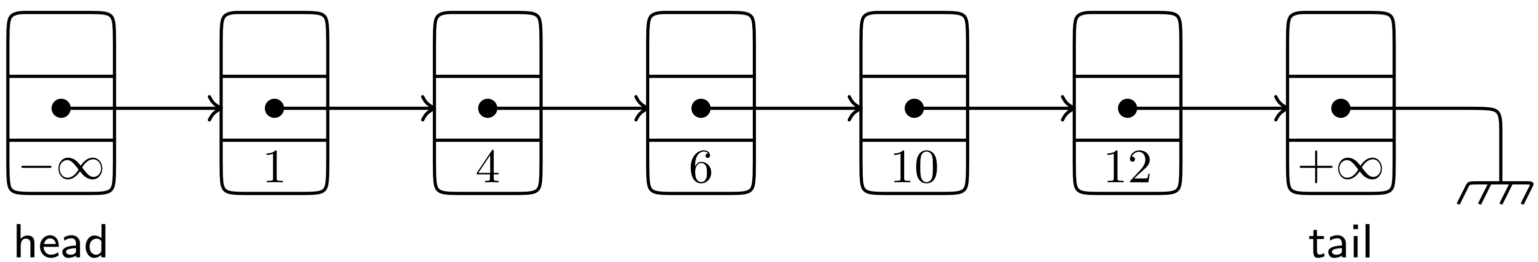
$$\text{list} \doteq \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\rightarrow}data = -\infty \wedge tail{\rightarrow}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \\ tail{\rightarrow}next = null \wedge tail \neq null \wedge head \neq null \wedge head \neq tail & \end{array}\right)$$



Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

$$\text{list} \hat{=} \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\rightarrow}data = -\infty \wedge tail{\rightarrow}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \\ tail{\rightarrow}next = null \wedge tail \neq null \wedge head \neq null \wedge head \neq tail & \end{array} \right)$$



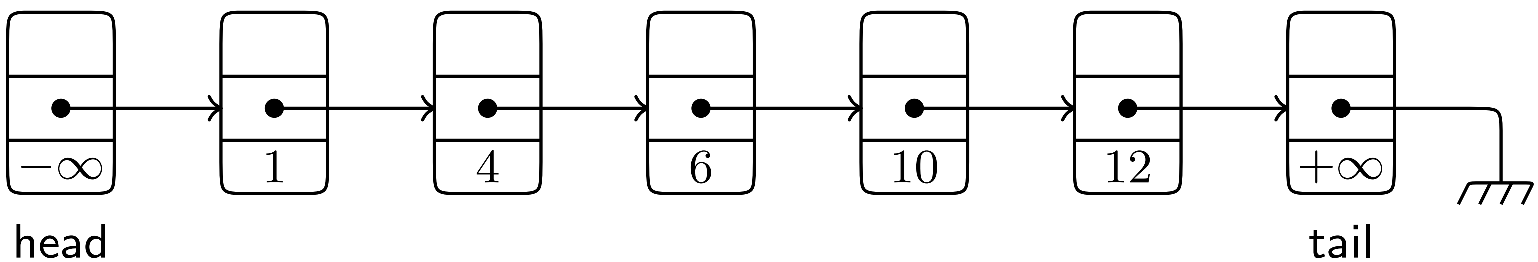
- We declare some **auxiliary invariants**

order $\rightsquigarrow$ "order is preserved"

Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

$$\text{list} \hat{=} \begin{pmatrix} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\rightarrow}data = -\infty \wedge tail{\rightarrow}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \\ tail{\rightarrow}next = null \wedge tail \neq null \wedge head \neq null \wedge head \neq tail & \end{pmatrix}$$



- We declare some **auxiliary invariants**

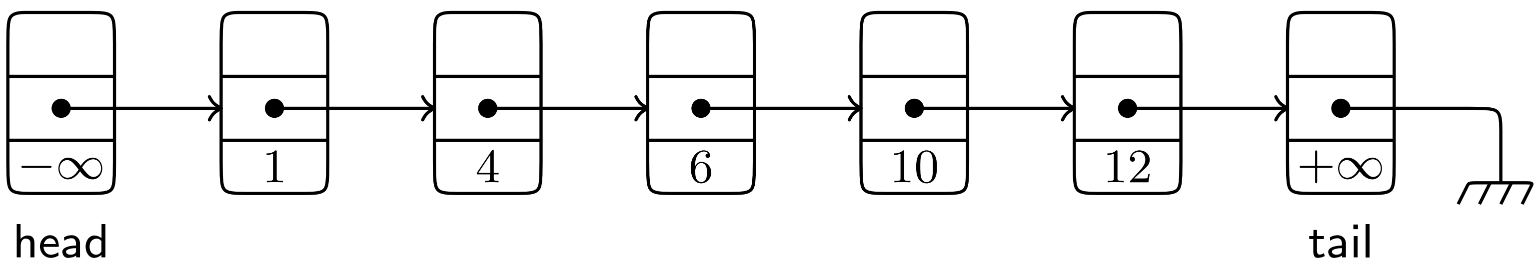
order $\rightsquigarrow$ "order is preserved"

next $\rightsquigarrow$ "pointers positions as they traverse the list"

Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

$$\text{list} \triangleq \left(\begin{array}{ll} reg = addr2set(heap, head) \land null \in reg & \land \\ head{\rightarrow}data = -\infty \land tail{\rightarrow}data = +\infty \land ordered(heap, head, tail) & \land \\ elems = set2elemset(heap, reg) & \land \\ tail{\rightarrow}next = null \land tail \neq null \land head \neq null \land head \neq tail & \end{array} \right)$$



- We declare some **auxiliary invariants**

order $\rightsquigarrow$ "order is preserved"

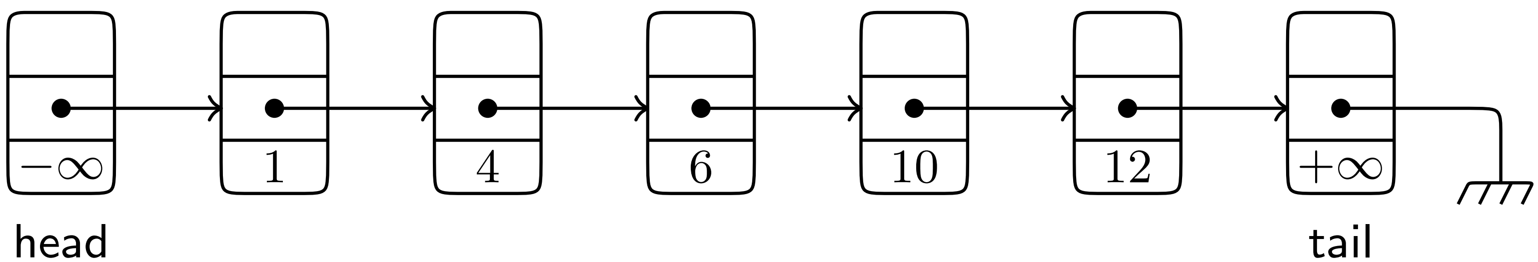
next $\rightsquigarrow$ "pointers positions as they traverse the list"

lock $\rightsquigarrow$ "tracking of locked nodes"

Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

$$\text{list} \mathrel{\hat{=}} \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\to}data = -\infty \wedge tail{\to}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \\ tail{\to}next = null \wedge tail \neq null \wedge head \neq null \wedge head \neq tail & \end{array}\right)$$



- We declare some **auxiliary invariants**

order ⇝ "order is preserved"

next ⇝ "pointers positions as they traverse the list"

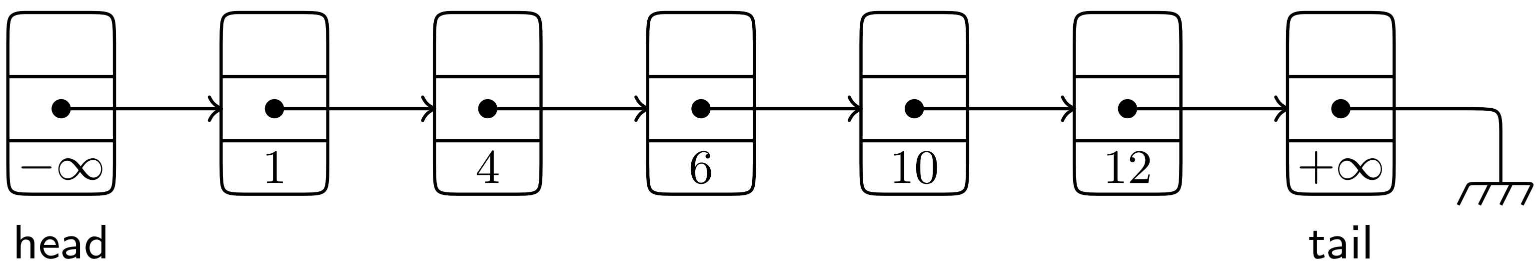
lock ⇝ "tracking of locked nodes"

region ⇝ "nodes within region"

Case Study 2: Lock-Coupling Lists

- We would like to verify list preservation (i.e., $\Box$list)

$$\text{list} \mathrel{\hat{=}} \left(\begin{array}{ll} reg = addr2set(heap, head) \wedge null \in reg & \wedge \\ head{\to}data = -\infty \wedge tail{\to}data = +\infty \wedge ordered(heap, head, tail) & \wedge \\ elems = set2elemset(heap, reg) & \wedge \\ tail{\to}next = null \wedge tail \neq null \wedge head \neq null \wedge head \neq tail & \end{array} \right)$$



- We declare some **auxiliary invariants**

order ⇝ "order is preserved"

next ⇝ "pointers positions as they traverse the list"

lock ⇝ "tracking of locked nodes"

region ⇝ "nodes within region"

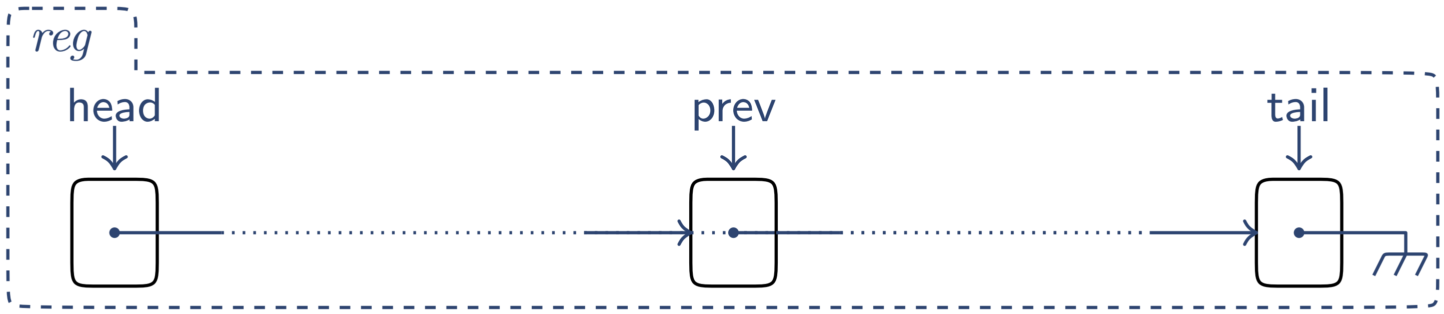
disj ⇝ "new nodes are disjoint from others"

DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region...

DEMO: lock-coupling lists (1st attempt)

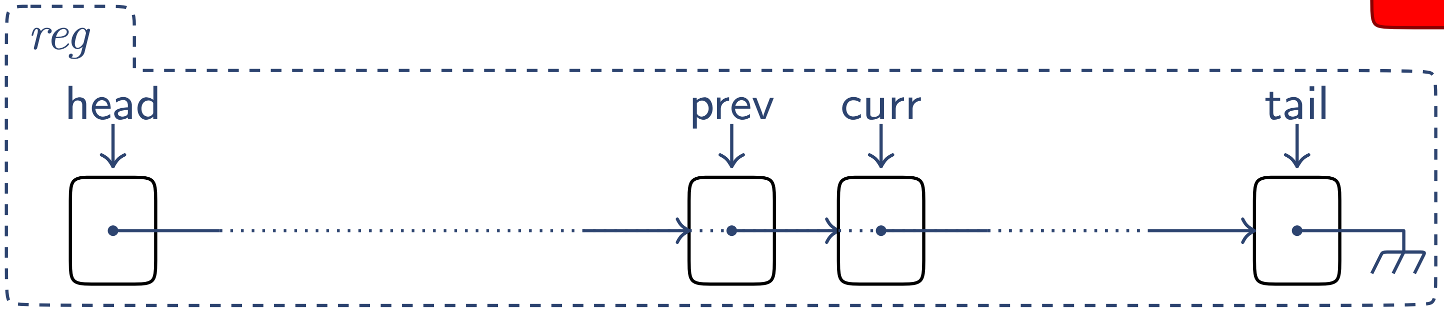
- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)



DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev{:=}\ curr$)

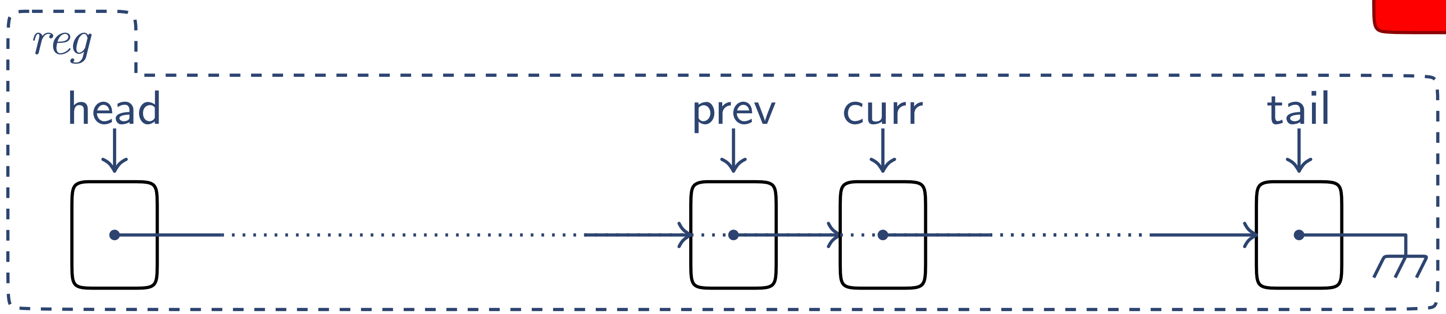
Need **next** as support



DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need **next** as support

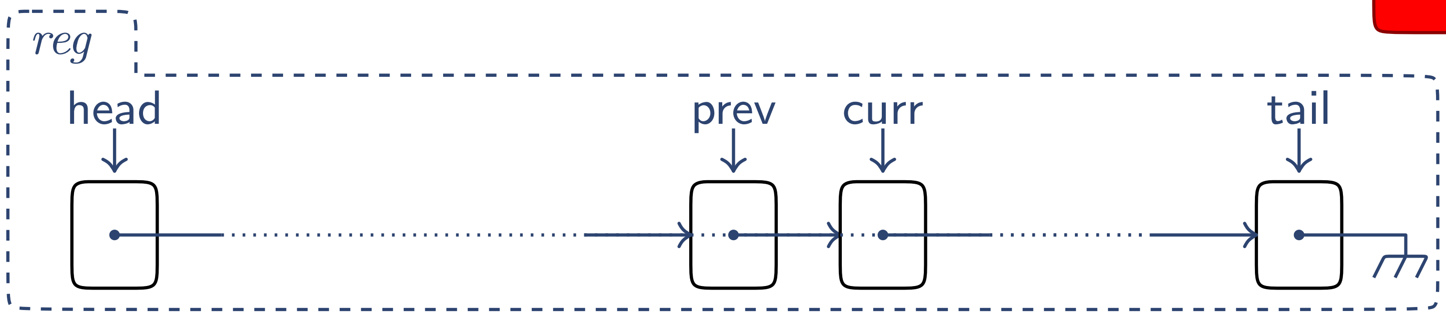


- And when verifying next...

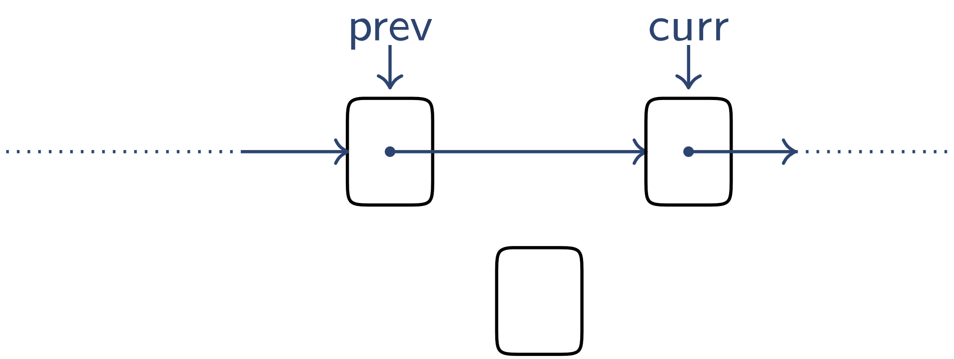
DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region…
- … **fails** on transition 28 (i.e., $prev := curr$)

Need **next** as support



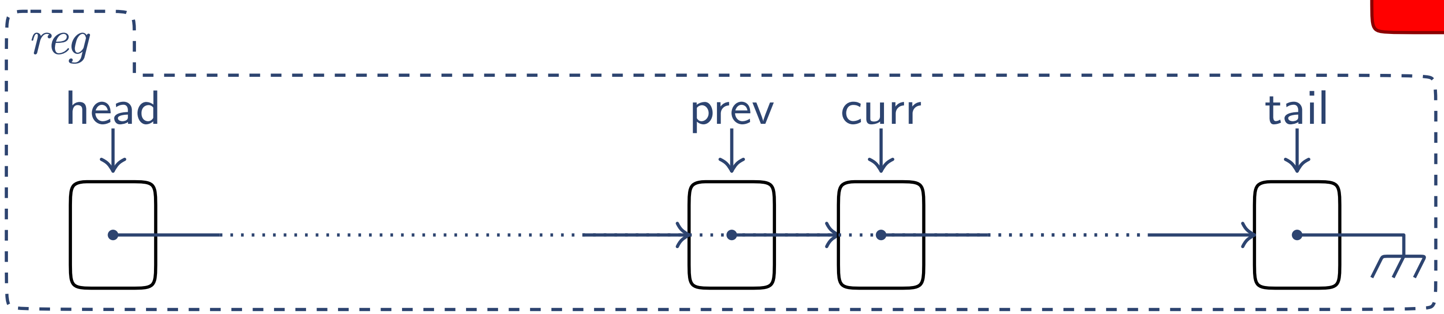
- And when verifying next…
- … **fails** on transition 34 (i.e., $aux \rightarrow next := curr$)



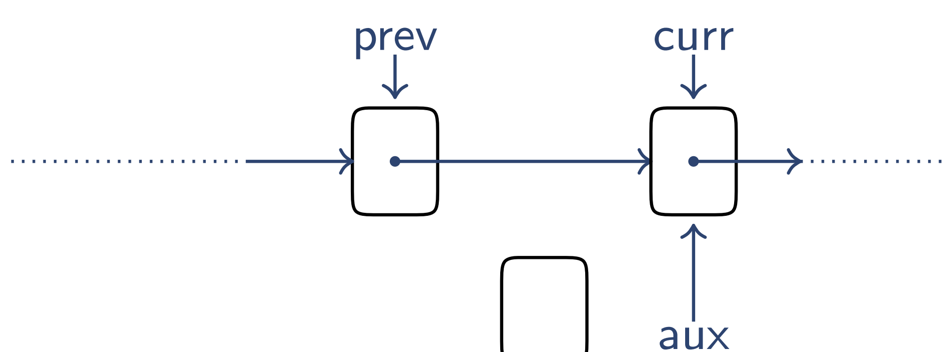
DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need **next** as support



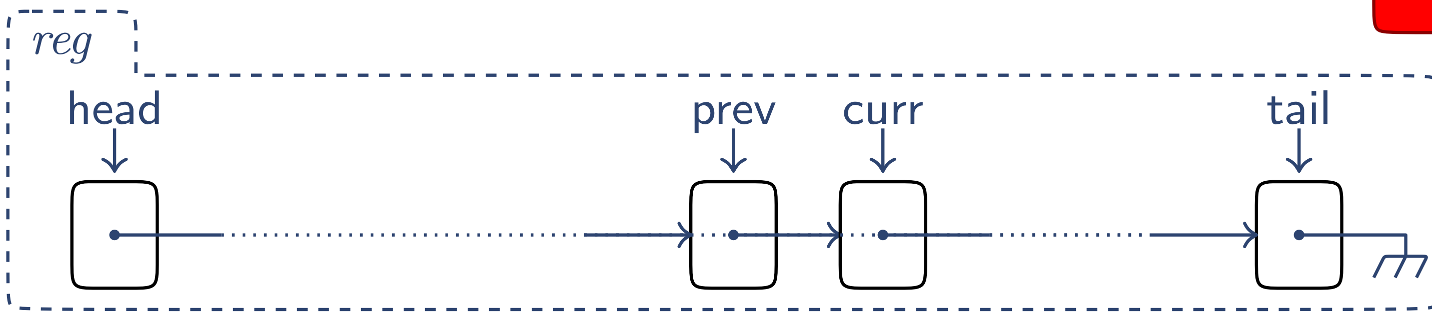
- And when verifying next...
- ... **fails** on transition 34 (i.e., $aux \to next := curr$)



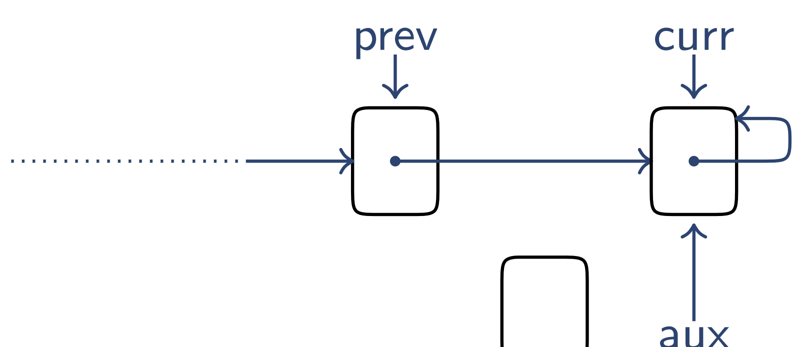
DEMO: lock-coupling lists (1st attempt)

- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need **next** as support



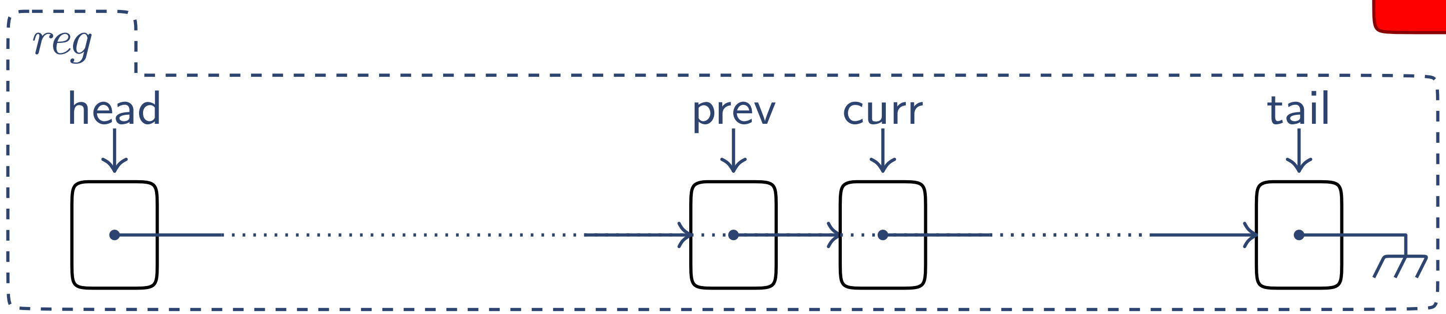
- And when verifying next...
- ... **fails** on transition 34 (i.e., $aux \rightarrow next := curr$)



DEMO: lock-coupling lists (1st attempt)

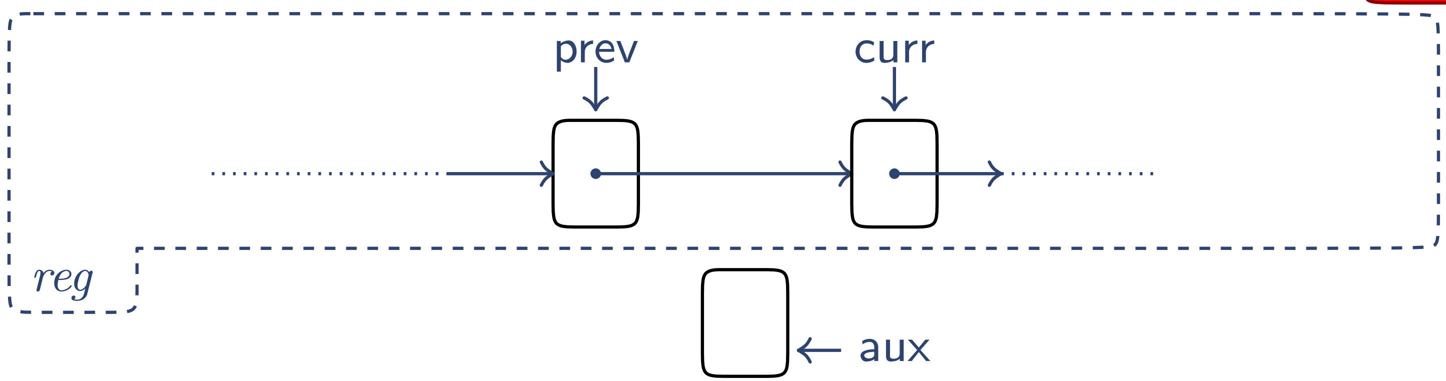
- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need next as support



- And when verifying next...
- ... **fails** on transition 34 (i.e., $aux \rightarrow next := curr$)

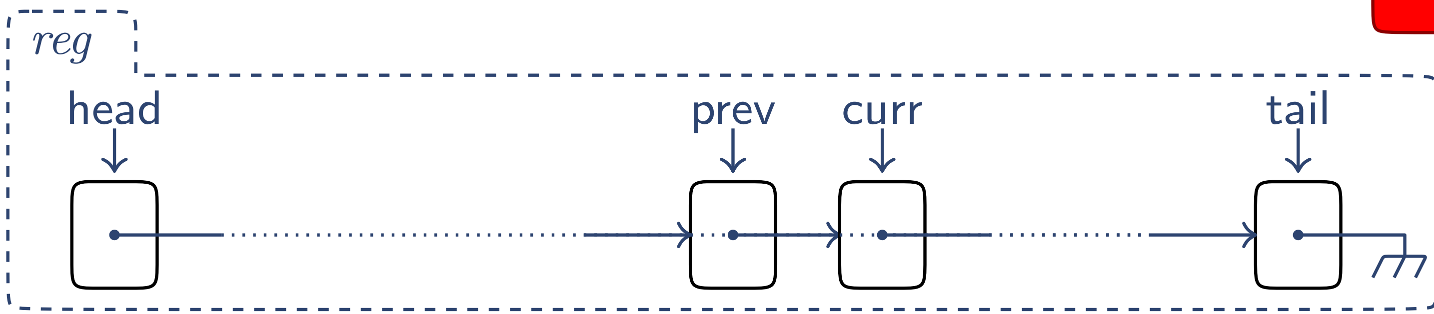
Need region as support



DEMO: lock-coupling lists (1st attempt)

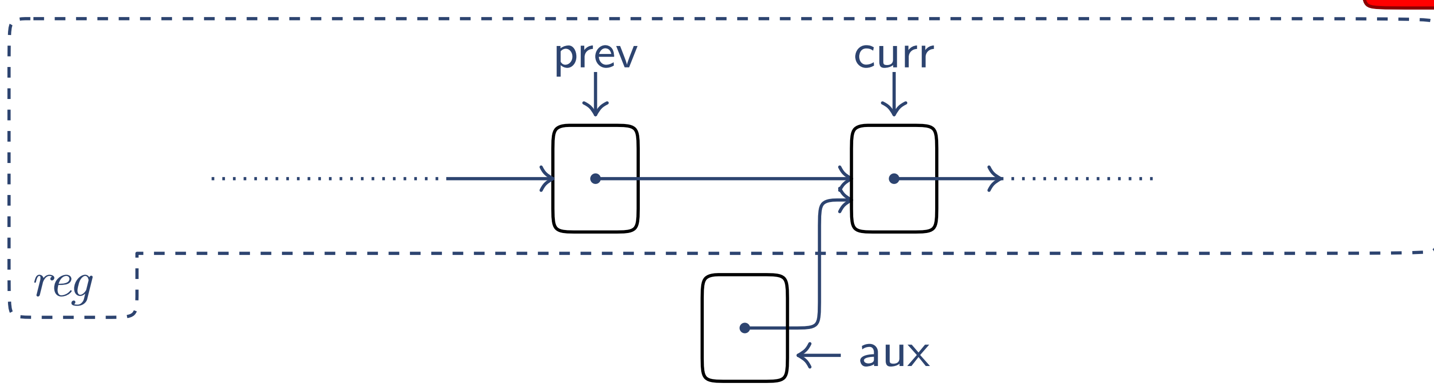
- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need next as support



- And when verifying next...
- ... **fails** on transition 34 (i.e., $aux \rightarrow next := curr$)

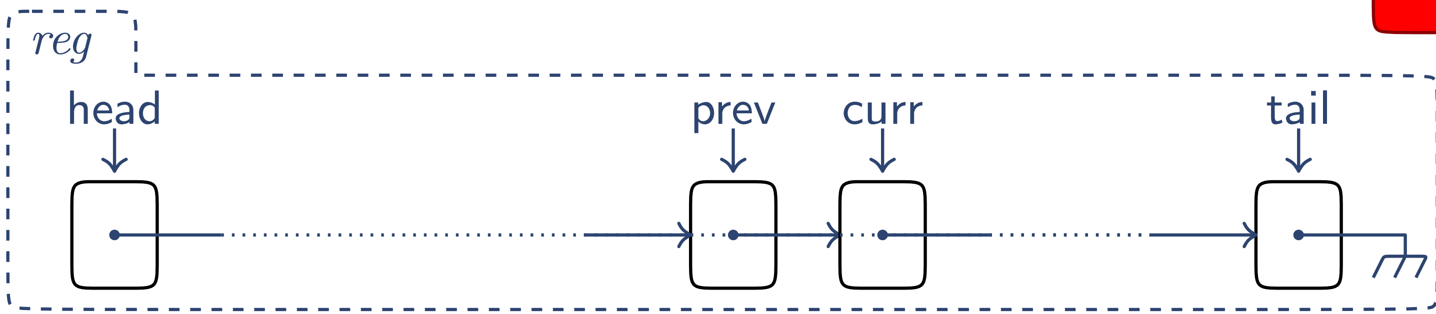
Need region as support



DEMO: lock-coupling lists (1st attempt)

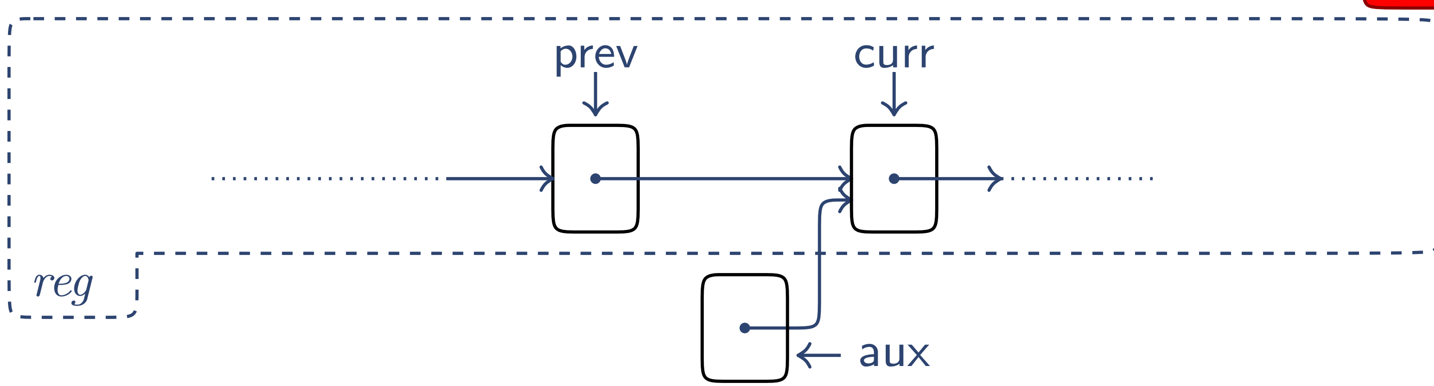
- Let's try to verify region...
- ... **fails** on transition 28 (i.e., $prev := curr$)

Need next as support



- And when verifying next...
- ... **fails** on transition 34 (i.e., $aux \rightarrow next := curr$)

Need region as support



We have **circularity**!

~~SP-INV~~

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
($\mathsf{SC_i}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
($\mathsf{SC_j}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
($\mathsf{OC_i}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
($\mathsf{OC_j}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$

$\Box\varphi_i \wedge \Box\varphi_j$

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

initiation

$$
\begin{array}{llcccl}
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi_i \wedge \varphi_j & \\
(\ \mathsf{SC_i}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ forall } t \in \overline{v} \\
(\ \mathsf{SC_j}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ forall } t \in \overline{w} \\
(\ \mathsf{OC_i}\ ) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
(\ \mathsf{OC_j}\ ) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ fresh } k \notin \overline{w} \\
\hline
 & & \Box\varphi_i \wedge \Box\varphi_j & & &
\end{array}
$$

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

initiation

self-consecution

(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
(SC$_i$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
(SC$_j$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
(OC$_i$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
(OC$_j$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$

$$\Box\varphi_i \wedge \Box\varphi_j$$

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
(SC$_i$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
(SC$_j$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
(OC$_i$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
(OC$_j$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$

$$\Box\varphi_i \wedge \Box\varphi_j$$

initiation

self-consecution

others-consecution

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

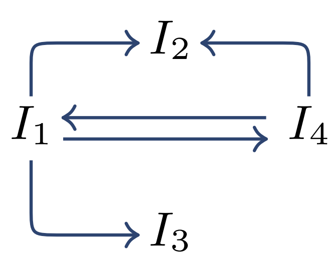
(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	**initiation**
($\mathsf{SC_i}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
($\mathsf{SC_j}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
($\mathsf{OC_i}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
($\mathsf{OC_j}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$

$$\Box\varphi_i \wedge \Box\varphi_j$$

self-consecution ($\mathsf{SC_i}$, $\mathsf{SC_j}$)

others-consecution ($\mathsf{OC_i}$, $\mathsf{OC_j}$)

- A **generalization** of G-INV is a **proof graph**



Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

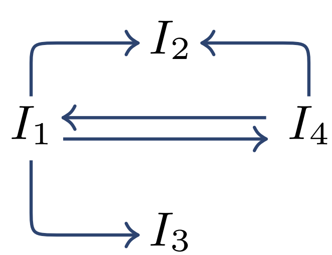
$$
\begin{array}{llrcll}
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi_i \wedge \varphi_j & \\
(\,\mathsf{SC_i}\,) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ forall } t \in \overline{v} \\
(\,\mathsf{SC_j}\,) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ forall } t \in \overline{w} \\
(\,\mathsf{OC_i}\,) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
(\,\mathsf{OC_j}\,) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ fresh } k \notin \overline{w} \\
\hline
\multicolumn{6}{c}{\Box\varphi_i \wedge \Box\varphi_j}
\end{array}
$$

initiation (I)

self-consecution ($\mathsf{SC_i}$, $\mathsf{SC_j}$)

others-consecution ($\mathsf{OC_i}$, $\mathsf{OC_j}$)

- A **generalization** of G-INV is a **proof graph**



Theorem

Every node is invariant if every node is either:

- an inductive invariant, or
- has an incident edge and all VCs are valid

Parametrized Invariance with Graph Support (g-inv)

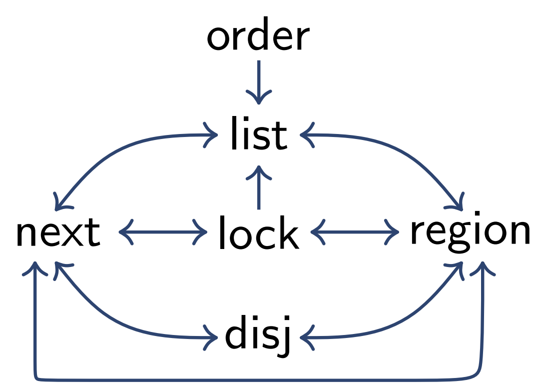
To show that $\mathcal{S}$ satisfies $\square\varphi_i(\overline{v}) \wedge \square\varphi_j(\overline{w})$:

$$
\begin{array}{llrcll}
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi_i \wedge \varphi_j & \\
(\,\mathsf{SC_i}\,) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ forall } t \in \overline{v} \\
(\,\mathsf{SC_j}\,) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ forall } t \in \overline{w} \\
(\,\mathsf{OC_i}\,) & \varphi_i, \varphi_j \rhd & \bigwedge\limits_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_i' & \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
(\,\mathsf{OC_j}\,) & \varphi_i, \varphi_j \rhd & \bigwedge\limits_{x \in \overline{w}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_j' & \text{forall } \tau, \text{ fresh } k \notin \overline{w} \\
\hline
\multicolumn{6}{c}{\square\varphi_i \wedge \square\varphi_j}
\end{array}
$$

initiation

self-consecution

others-consecution



Theorem

Every node is invariant if every node is either:

- an inductive invariant, or
- has an incident edge and all VCs are valid

Experimental Results

	formula info		#solved vc		FS+TA	FS+RS	Graph
	index	#vc	pos	dp	time(s)	time(s)	time(s)
list	0	61	19	42	146.1	34.9	3.9
order	1	121	38	83	107.6	91.4	5.3
lock	1	121	57	64	39.7	8.9	1.8
next	1	121	38	83	166.7	18.4	3.4
region	1	121	95	26	15.7	4.1	1.3
disj	2	181	177	4	80.6	5.1	0.6
mutex	2	28	26	2	0.2	0.2	0.1
minticket	1	19	18	1	0.2	0.2	0.1
notsame	2	28	25	3	0.2	0.1	0.1
activelow	1	19	17	2	0.1	0.1	0.1

Experimental Results

	formula info		#solved vc		FS+TA time(s)	FS+RS time(s)	Graph time(s)
	index	#vc	pos	dp			
list	0	61	19	42	146.1	34.9	3.9
order	1	121	38	83	107.6	91.4	5.3
lock	1	121	57	64	39.7	8.9	1.8
next	1	121	38	83	166.7	18.4	3.4
region	1	121	95	26	15.7	4.1	1.3
disj	2	181	177	4	80.6	5.1	0.6
mutex	2	28	26	2	0.2	0.2	0.1
minticket	1	19	18	1	0.2	0.2	0.1
notsame	2	28	25	3	0.2	0.1	0.1
activelow	1	19	17	2	0.1	0.1	0.1

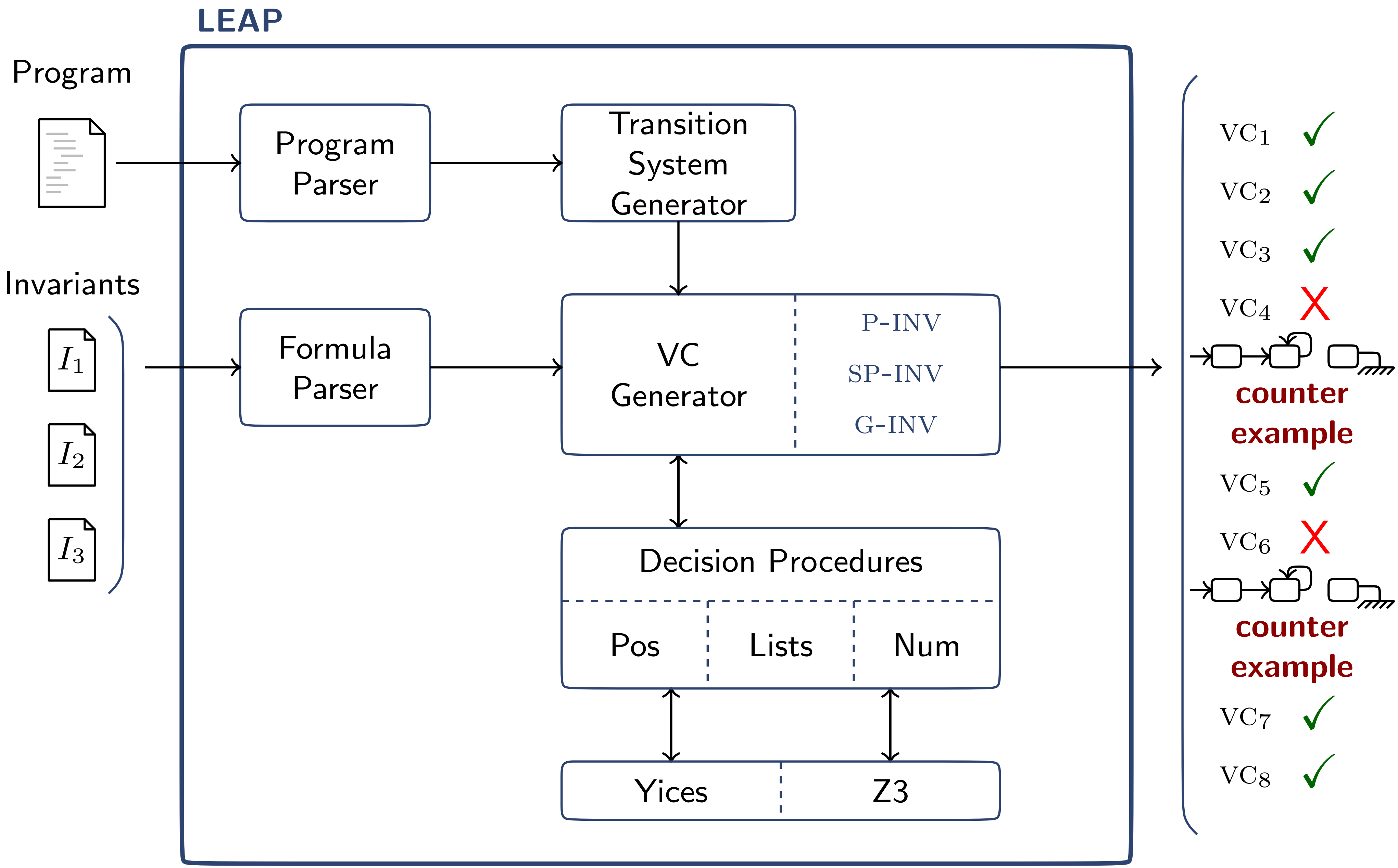
Mutual exclusion algorithm: 94 VCs / 0.4 sec.

Experimental Results

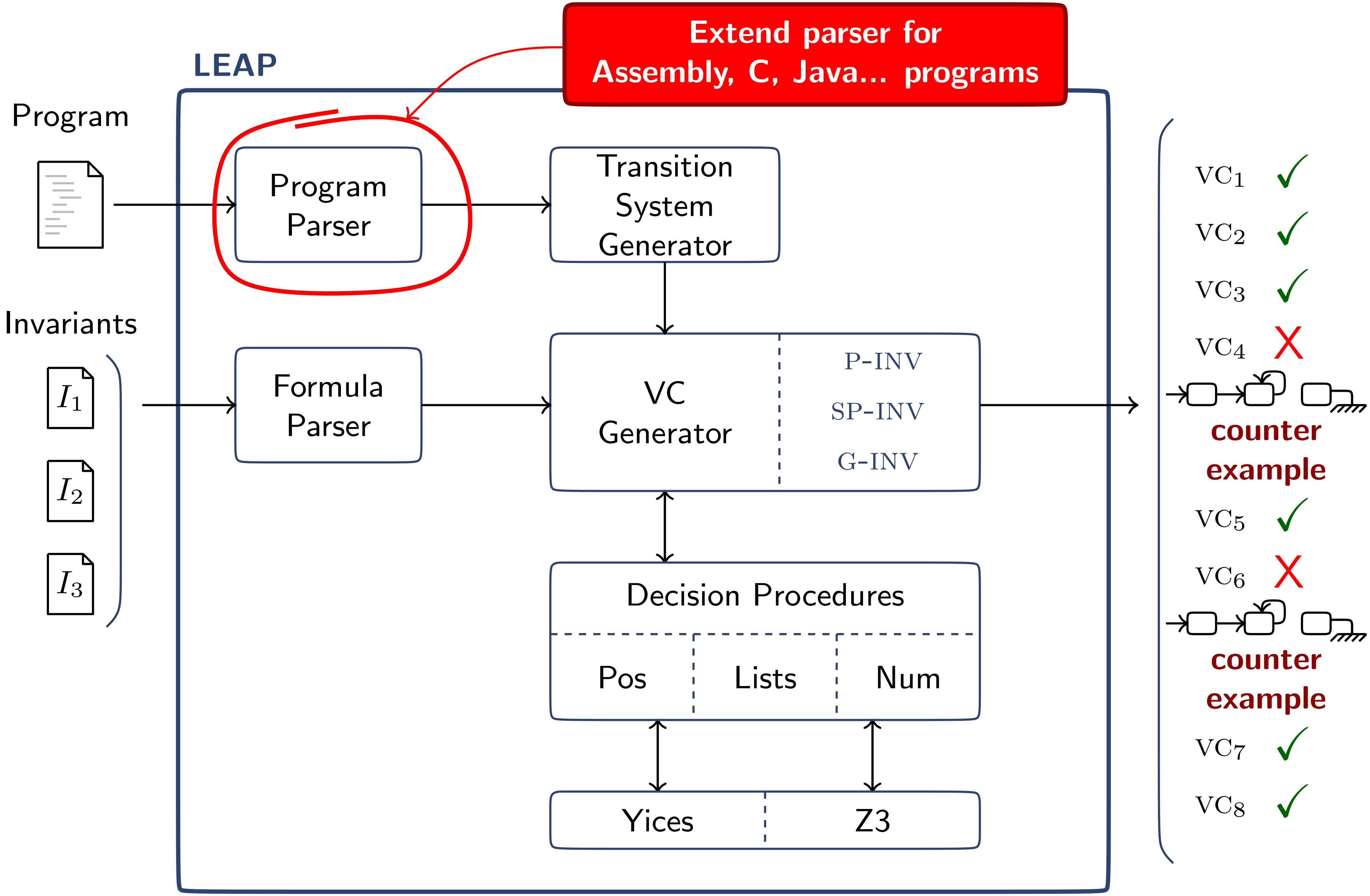
	formula info		#solved vc		FS+TA time(s)	FS+RS time(s)	Graph time(s)
	index	#vc	pos	dp			
list	0	61	19	42	146.1	34.9	3.9
order	1	121	38	83	107.6	91.4	5.3
lock	1	121	57	64	39.7	8.9	1.8
next	1	121	38	83	166.7	18.4	3.4
region	1	121	95	26	15.7	4.1	1.3
disj	2	181	177	4	80.6	5.1	0.6
mutex	2	28	26	2	0.2	0.2	0.1
minticket	1	19	18	1	0.2	0.2	0.1
notsame	2	28	25	3	0.2	0.1	0.1
activelow	1	19	17	2	0.1	0.1	0.1

Lock-coupling Lists: 726 VCs / 16.3 sec.

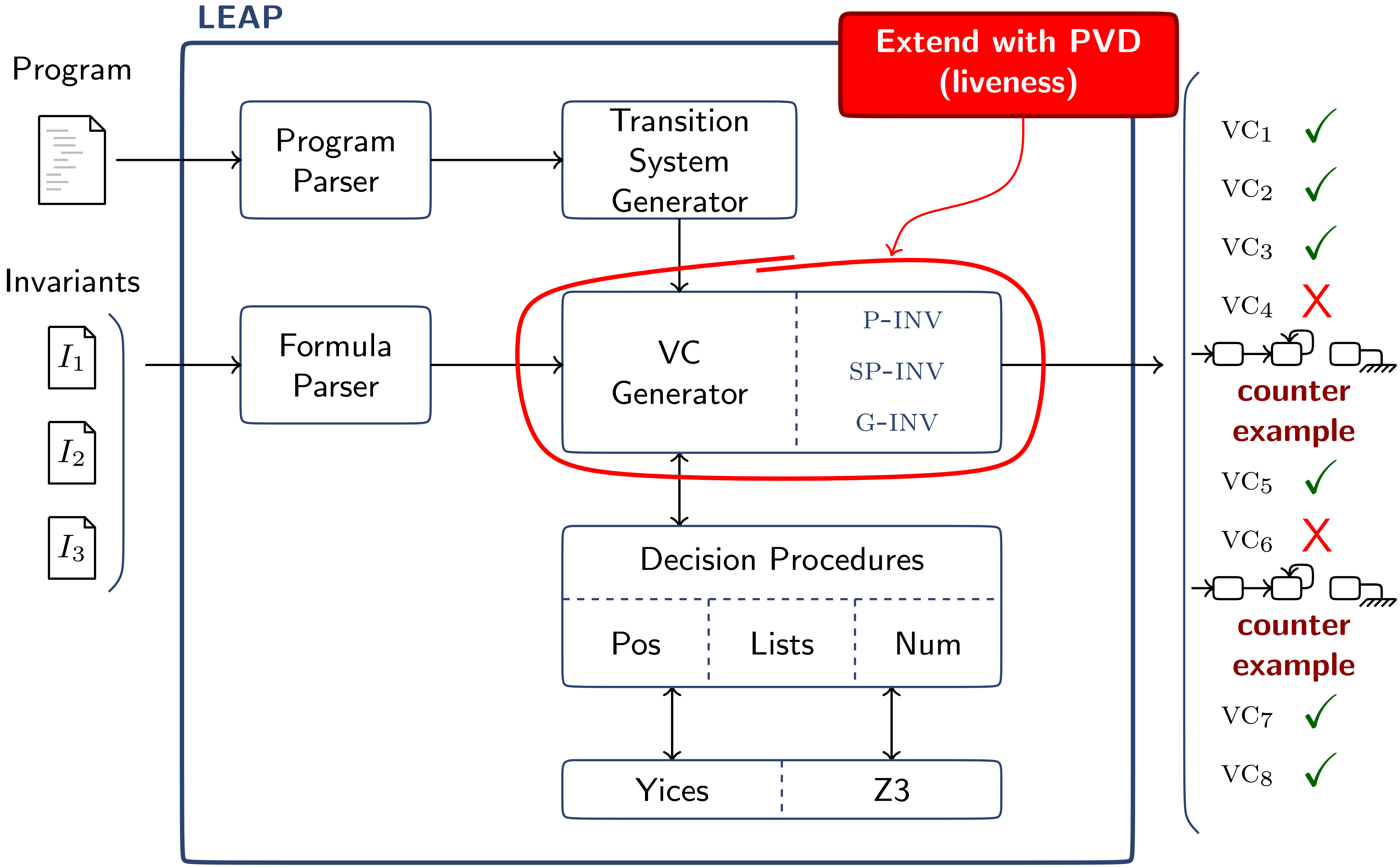
Future Ideas



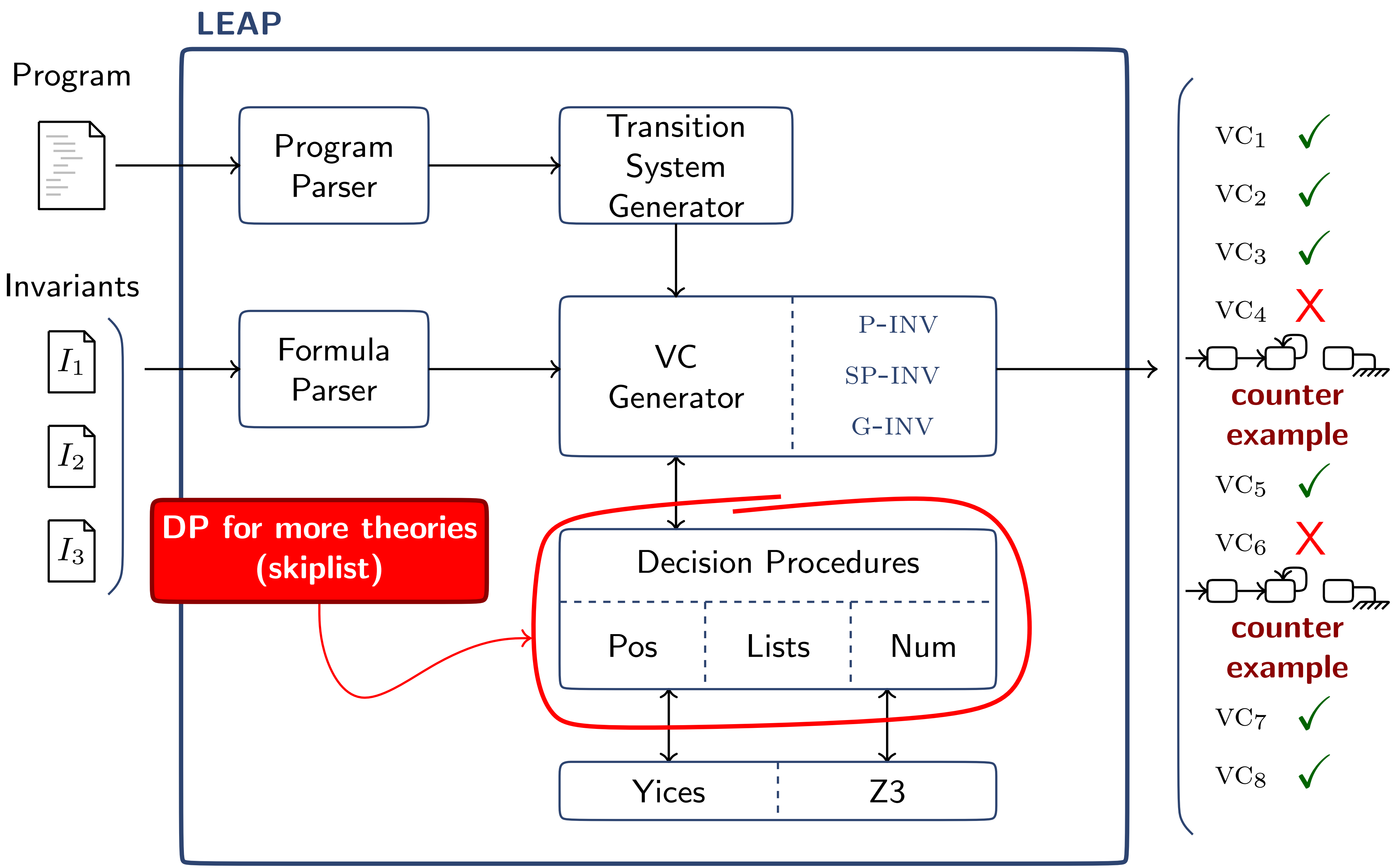
Future Ideas



Future Ideas

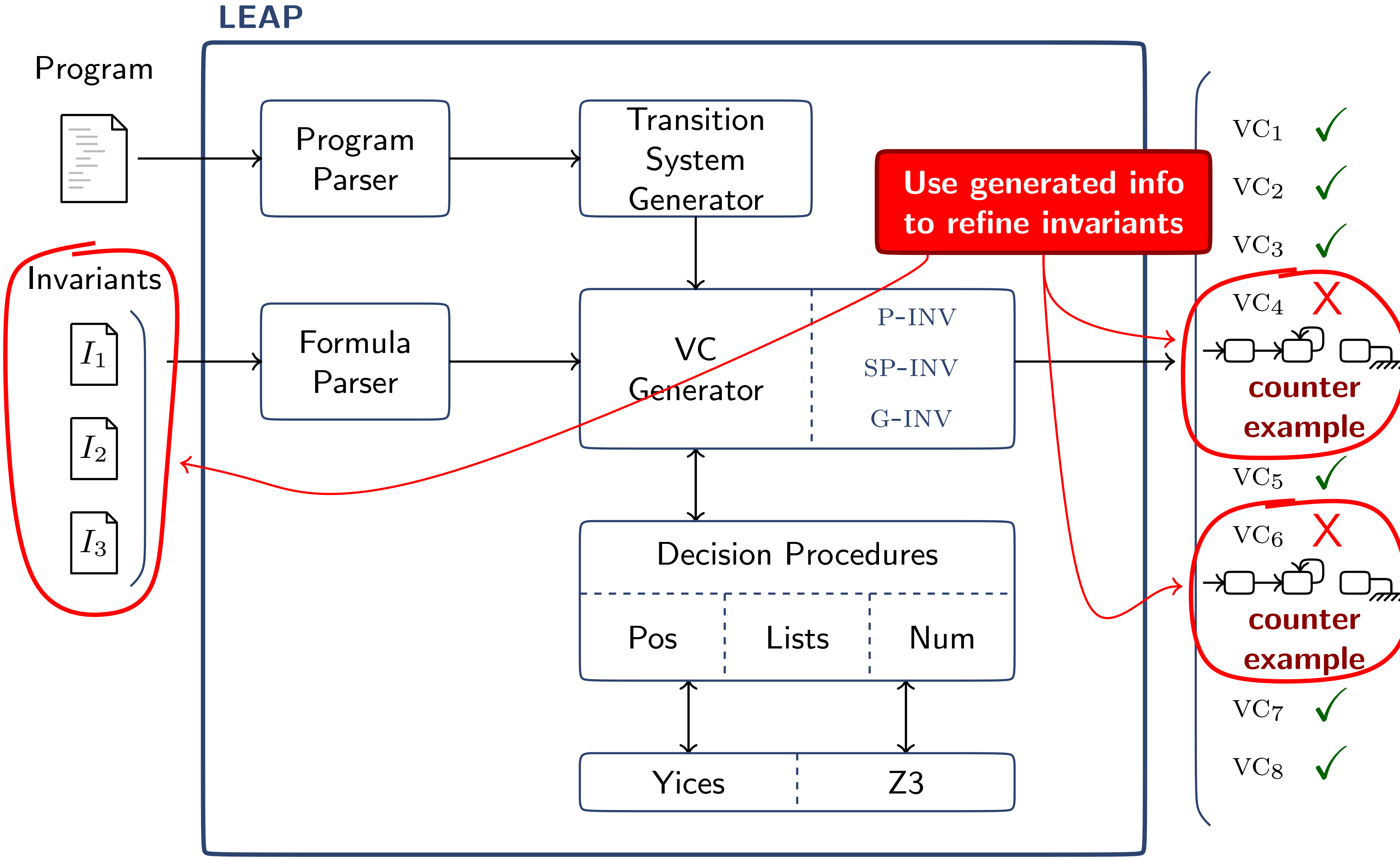


Future Ideas

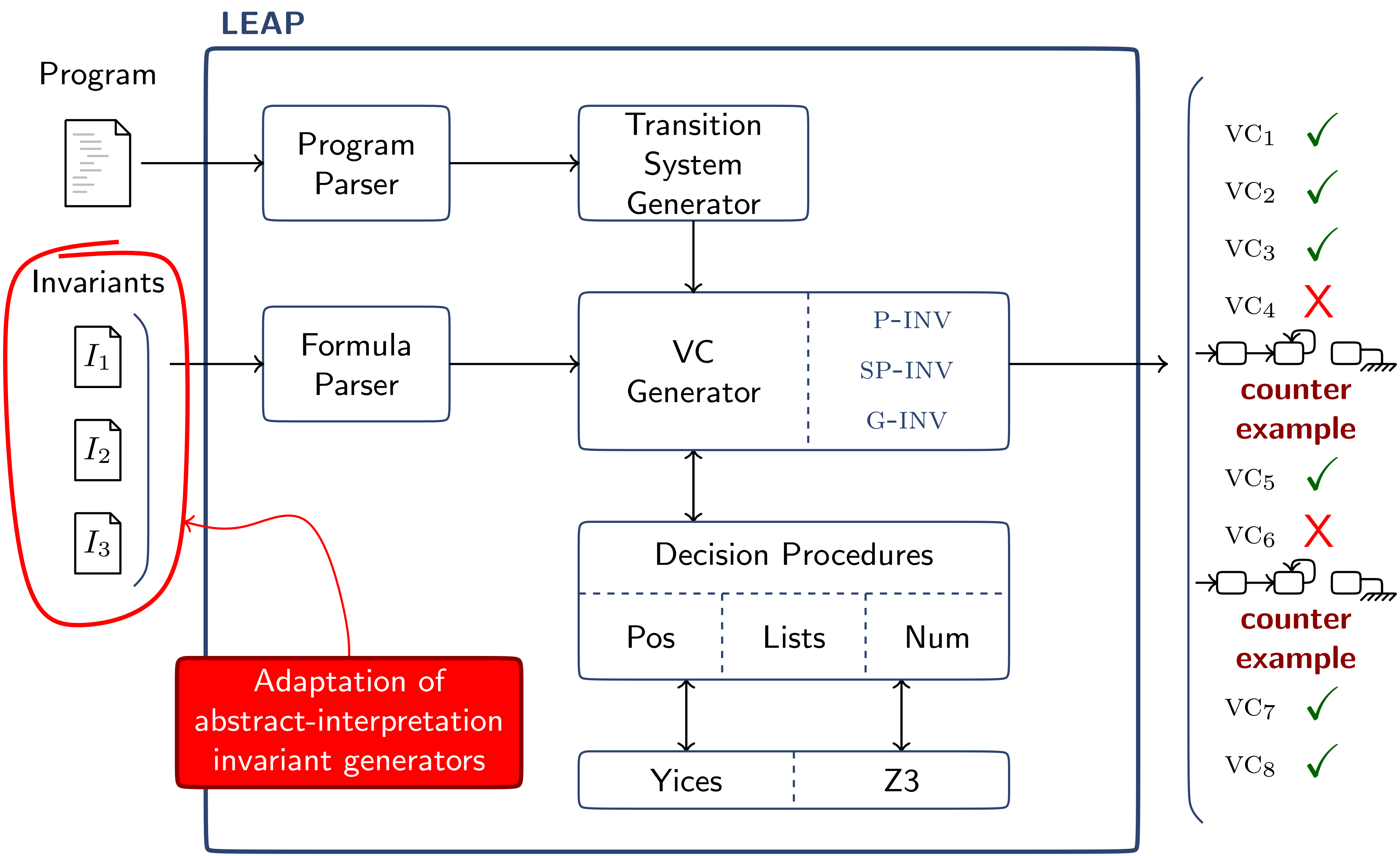


Future Ideas

refine supoported invariants



Future Ideas



Conclusions

- We presented a technique for verifying **parametrized invariance**
- Requires verification of a **bounded number of VCs**...
- ... **independently** of number of **threads**
- **Implemented** on LEAP
- **Easily extensible** for other **languages and theories**
- Uniform verification of **safety properties**...
- ... a step **towards liveness verification**