Decision Procedures for the Temporal Verification of Concurrent Lists

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

[1]The IMDEA Software Institute, Spain

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

ICFEM'10, Shanghai, 17 November 2010

Motivation

Why do we need decision procedures for concurrent data structures?

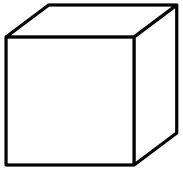
Verification of Concurrent Data-structures

Main Idea

Verification of Concurrent Data-structures

Main Idea

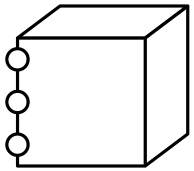
Concurrent DataStructure



Verification of Concurrent Data-structures

Main Idea

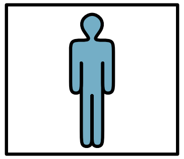
Concurrent DataStructure

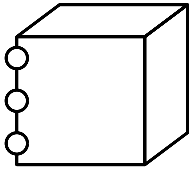


Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



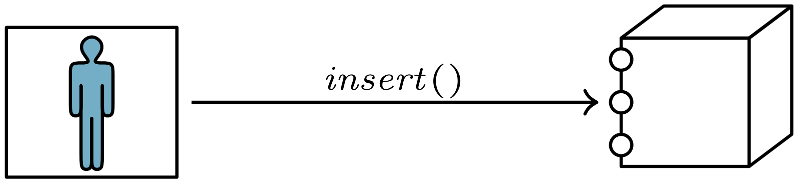


Most General Client

Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure

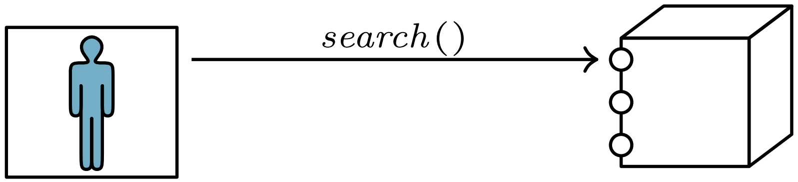


Most General Client

Verification of Concurrent Data-structures

Main Idea

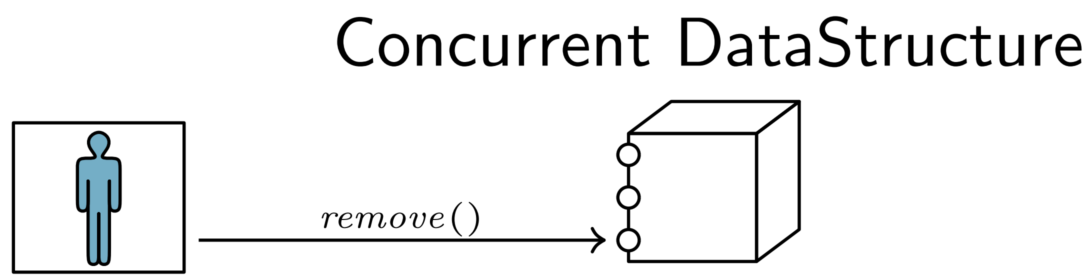
Concurrent DataStructure



Most General Client

Verification of Concurrent Data-structures

Main Idea



Most General Client

Verification of Concurrent Data-structures

Main Idea

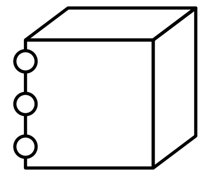
Concurrent DataStructure

Most General Client

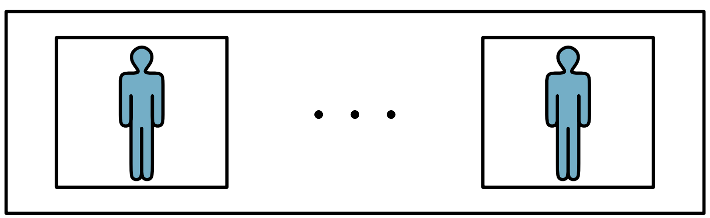
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



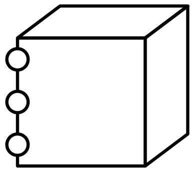
Most General Client



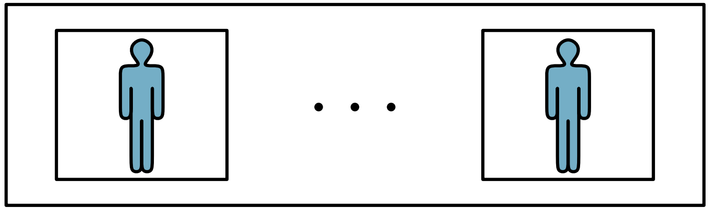
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client

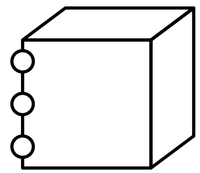


$$P[N] : P(1) || \cdots || P(N)$$

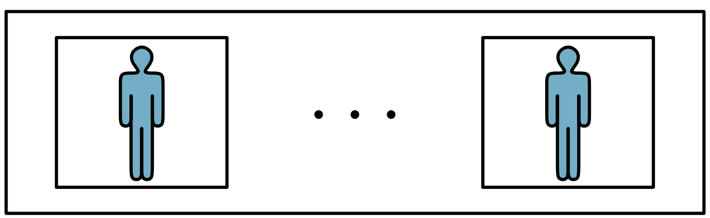
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$$P[N] : P(1) || \cdots || P(N)$$

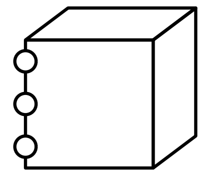
$$+$$

ghost variables

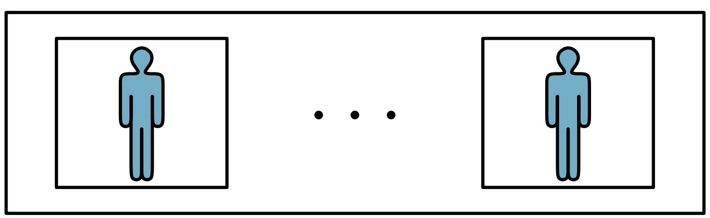
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$$P[N] : P(1) || \cdots || P(N)$$

$$+$$

ghost variables

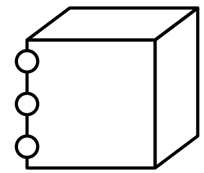
Property

$$\varphi^{(k)}$$

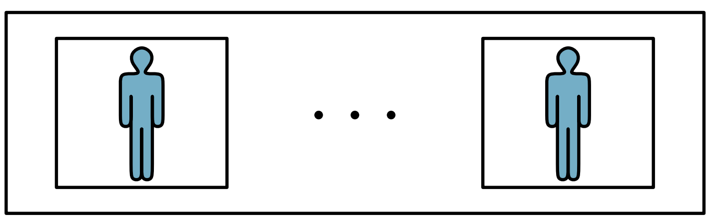
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$$P[N] : P(1) || \cdots || P(N)$$

$$+$$

ghost variables

Property

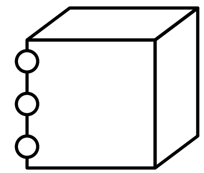
$$\varphi^{(k)}$$

LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

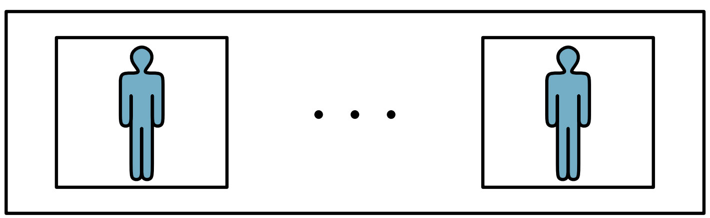
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1) || \cdots || P(N)$

$+$

ghost variables

Diagram

$\mathcal{D}$

Property

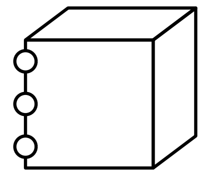
$\varphi^{(k)}$

LTL ($\Box$, $\Diamond$, $\mathcal{U}$, ...)

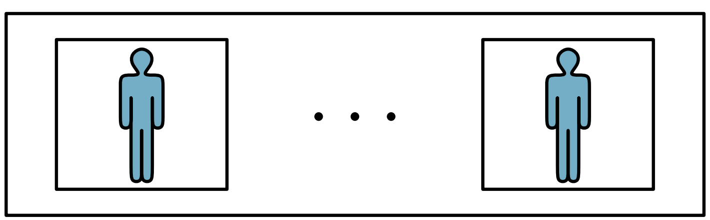
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client

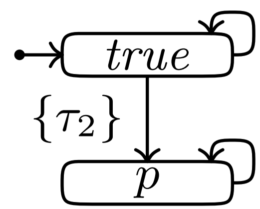


$$P[N] : P(1)|| \cdots ||P(N)$$

$+$

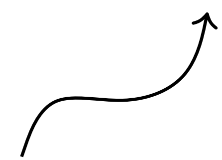
ghost variables

Diagram



Property

$\varphi^{(k)}$

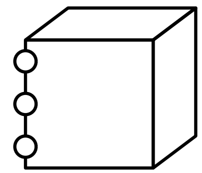


LTL ($\Box$,$\Diamond$,$\mathcal{U}$,$\ldots$)

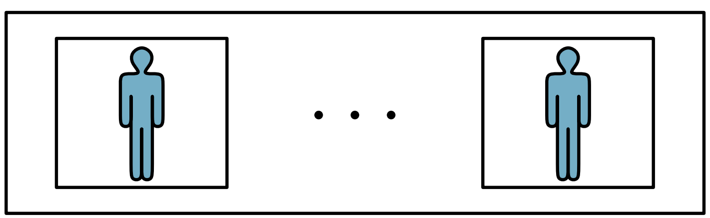
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$$P[N] : P(1) || \cdots || P(N)$$

$$+$$

ghost variables

Diagram

$$\mathcal{D}$$

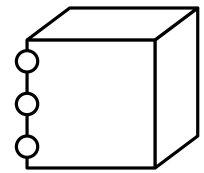
Property

$$\varphi^{(k)}$$

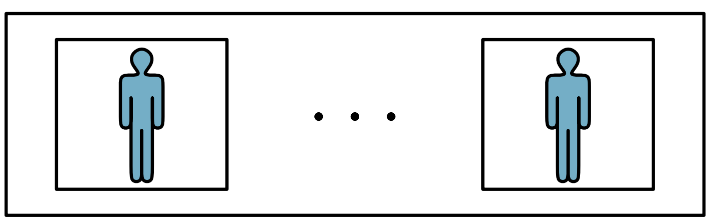
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1) || \cdots || P(N)$

$+$

ghost variables

Diagram

Property

$\models \quad \mathcal{D} \quad \models \quad \varphi^{(k)}$

Verification Conditions:

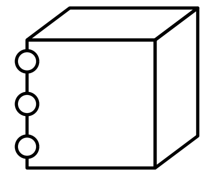
- Initiation
- Consecution
- Acceptance
- Fairness

Satisfaction (Model Checking)

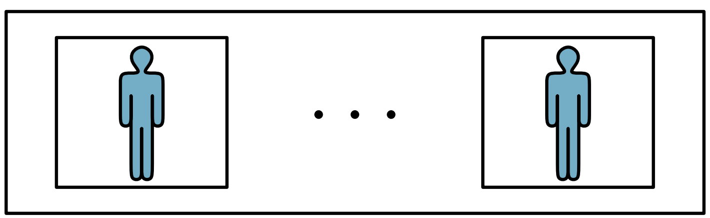
Verification of Concurrent Data-structures

Main Idea

Concurrent DataStructure



Most General Client



$P[N] : P(1) || \cdots || P(N)$

$+$

ghost variables

Diagram

Property

$\models \quad \mathcal{D} \quad \models \quad \varphi^{(k)}$

Verification Conditions:

- Initiation
- Consecution
- Acceptance
- Fairness

Satisfaction
(Model Checking)

Decision Procedures
(first order propositional logic)

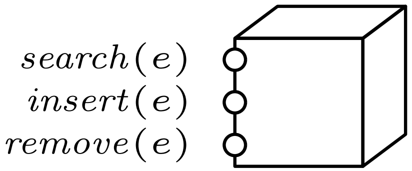
Concurrent Lock-coupling Lists

Concurrent Lock-coupling Lists

- A concurrent implementation of sets

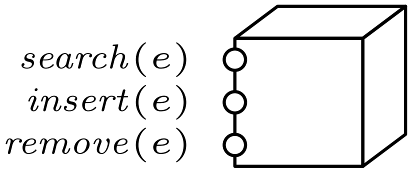
Concurrent Lock-coupling Lists

- A concurrent implementation of sets



Concurrent Lock-coupling Lists

- A concurrent implementation of sets

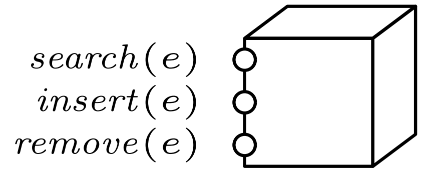



```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```

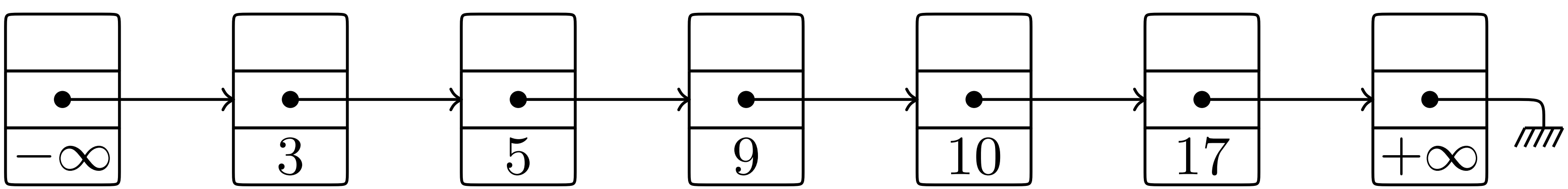
Concurrent Lock-coupling Lists

- A concurrent implementation of sets



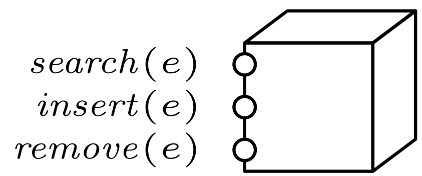

```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



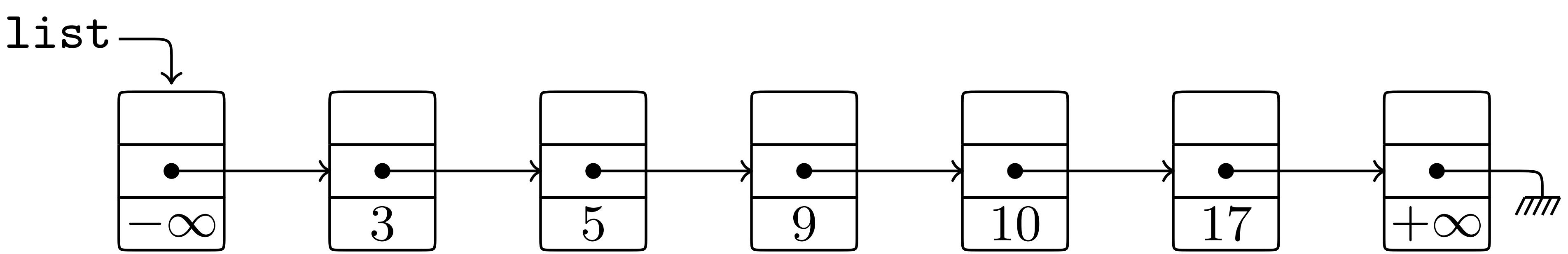
Concurrent Lock-coupling Lists

- A concurrent implementation of sets



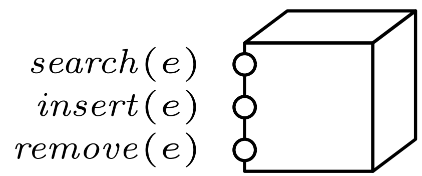

```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



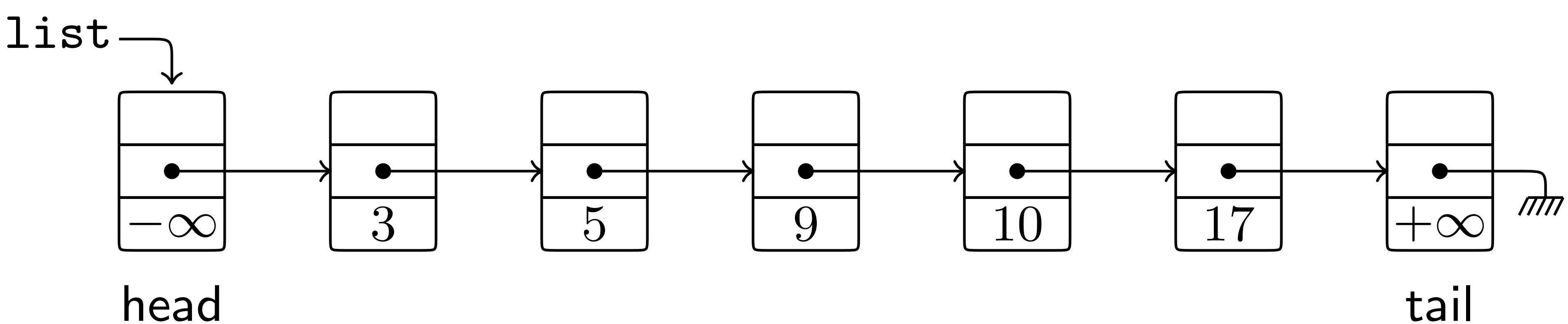
Concurrent Lock-coupling Lists

- A concurrent implementation of sets



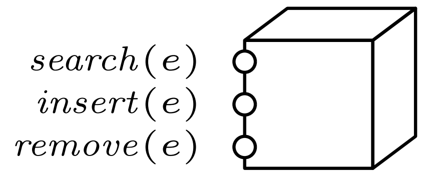

```
class Node {
      Value val;
      Node next;
      Lock lock;
}

class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

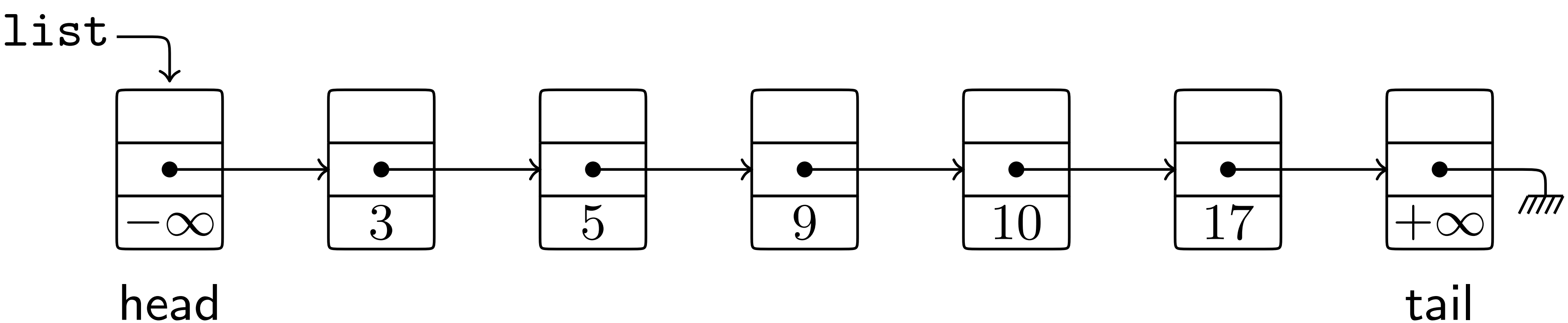
- A concurrent implementation of sets



search(8)

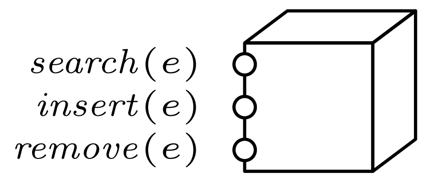
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

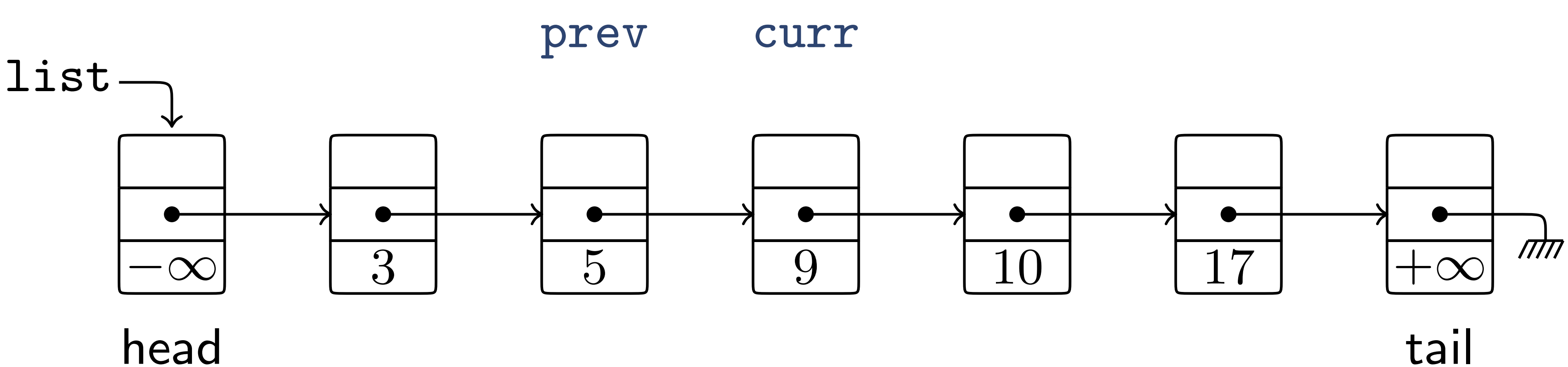
- A concurrent implementation of sets



search(8)

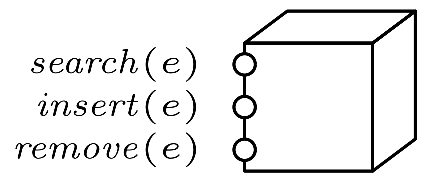
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

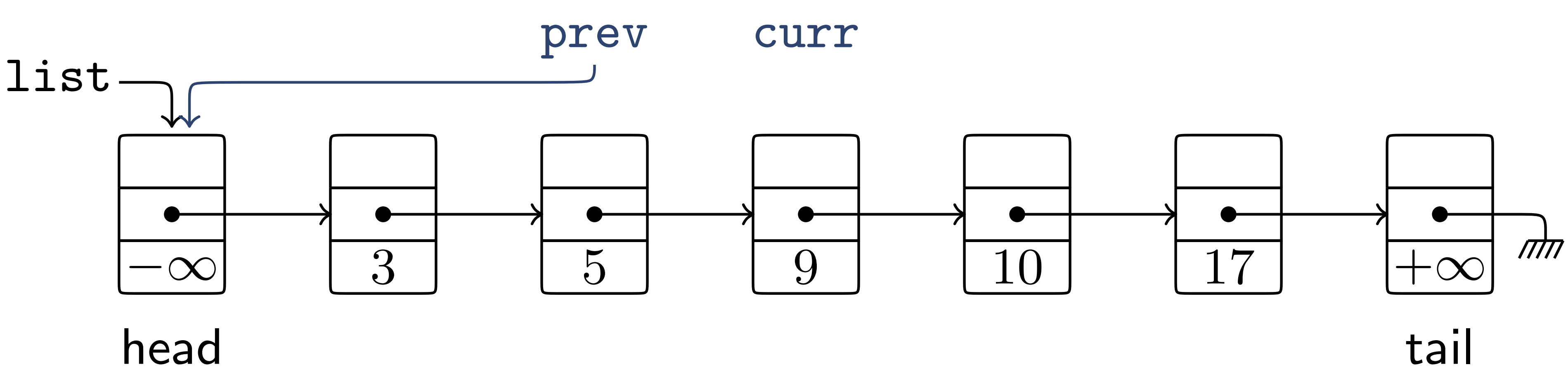
- A concurrent implementation of sets



$search(8)$

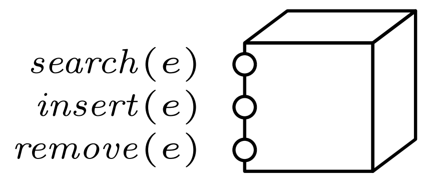
```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



Concurrent Lock-coupling Lists

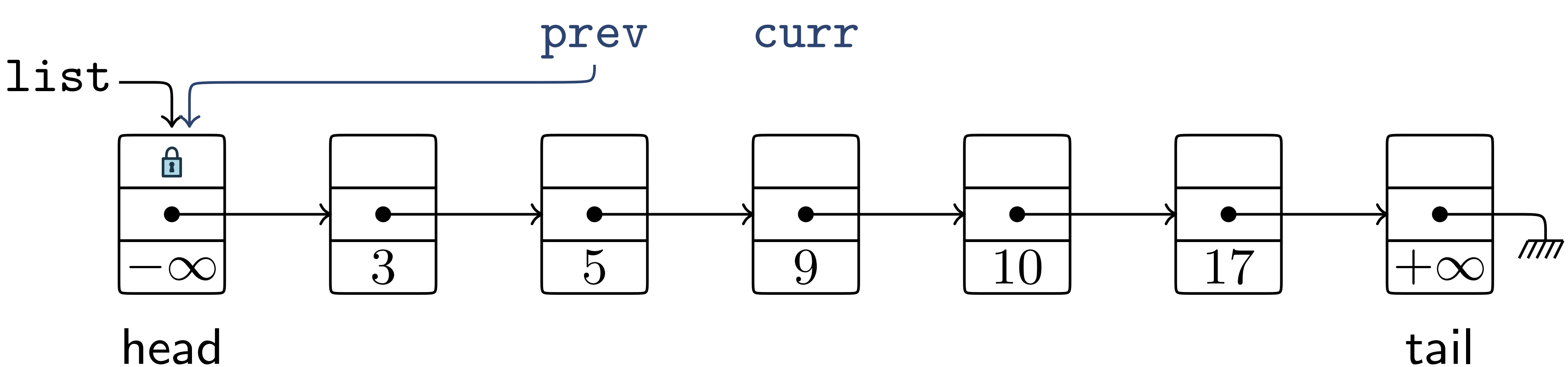
- A concurrent implementation of sets



search(8)

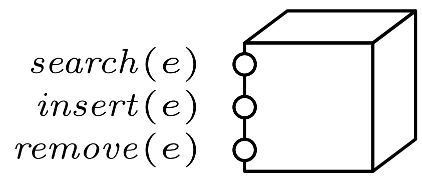
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

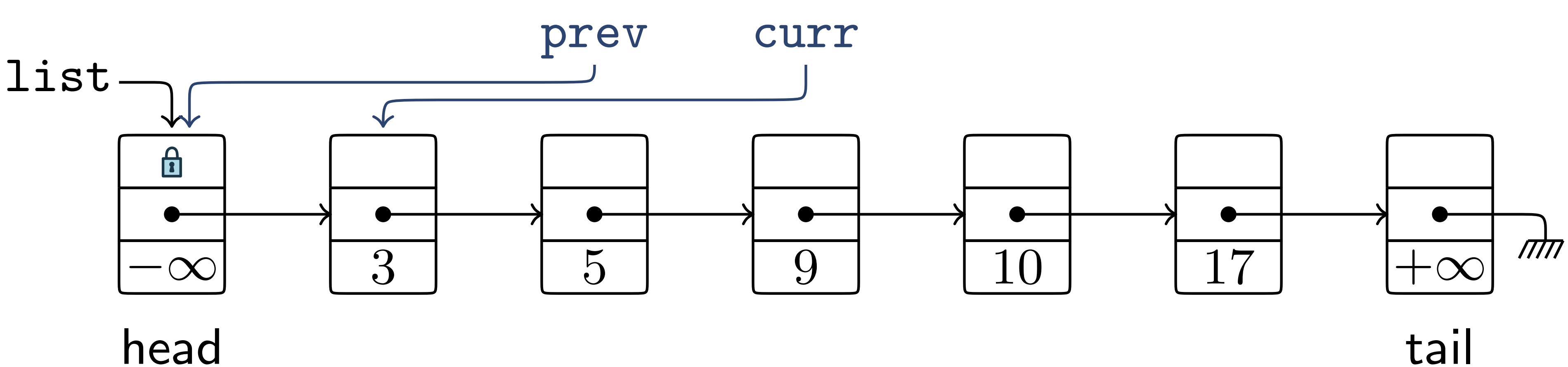
- A concurrent implementation of sets



$search(8)$

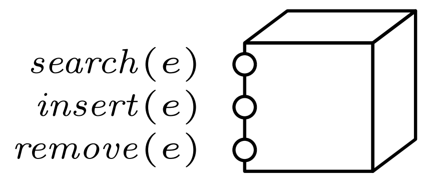
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

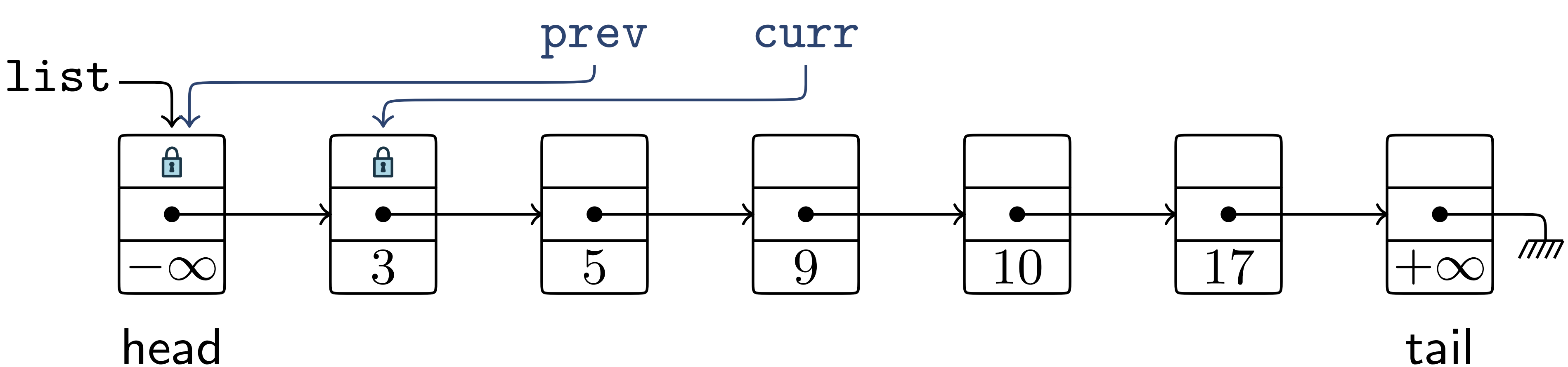
- A concurrent implementation of sets



search(8)

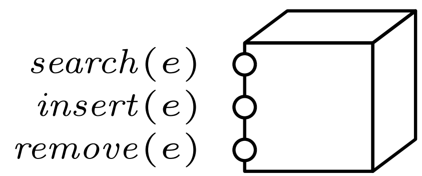
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

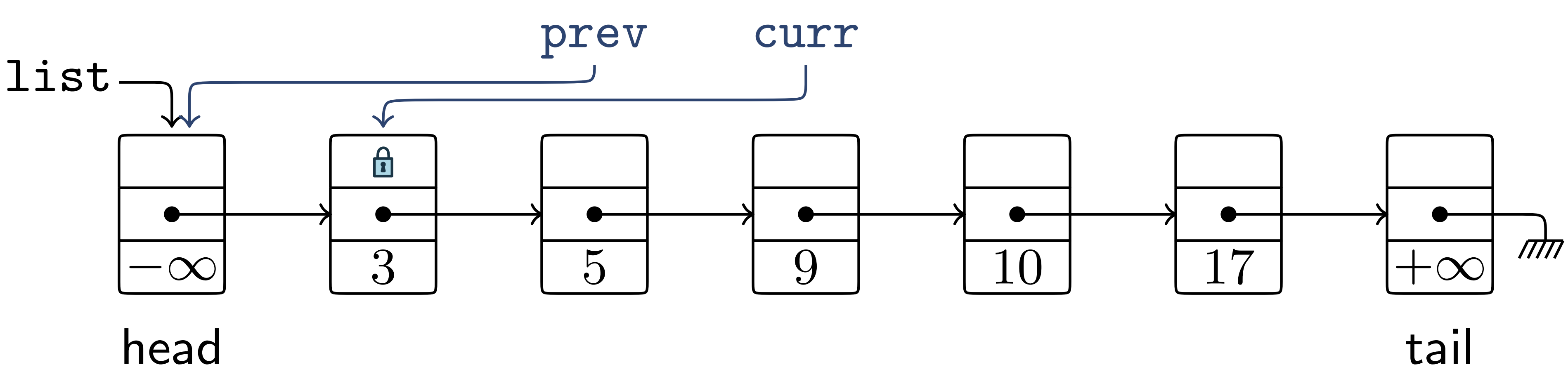
- A concurrent implementation of sets



search(8)

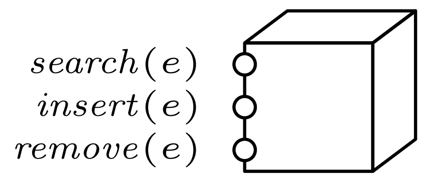
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

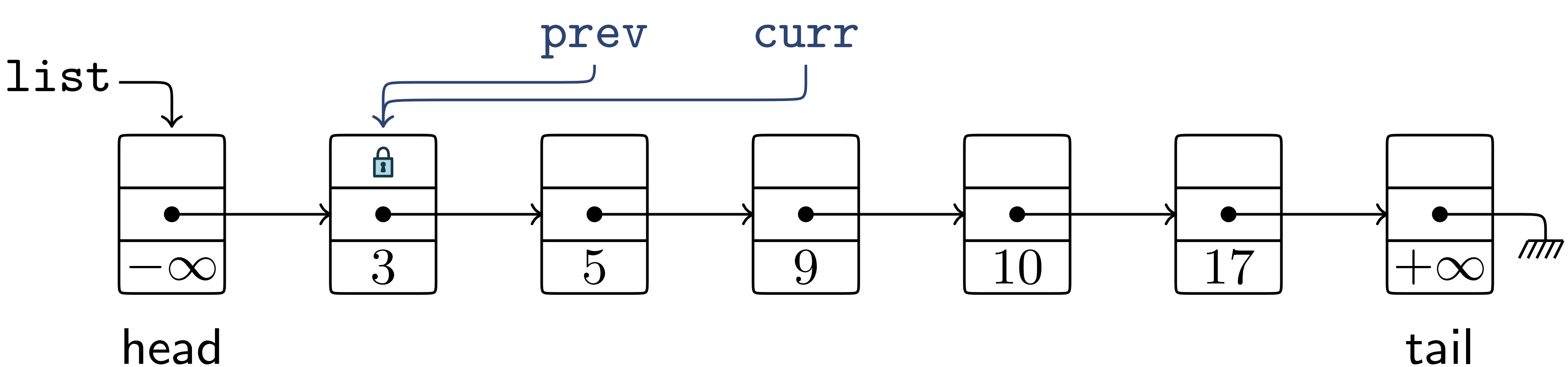
- A concurrent implementation of sets



search(8)

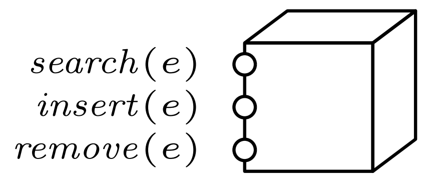
```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



Concurrent Lock-coupling Lists

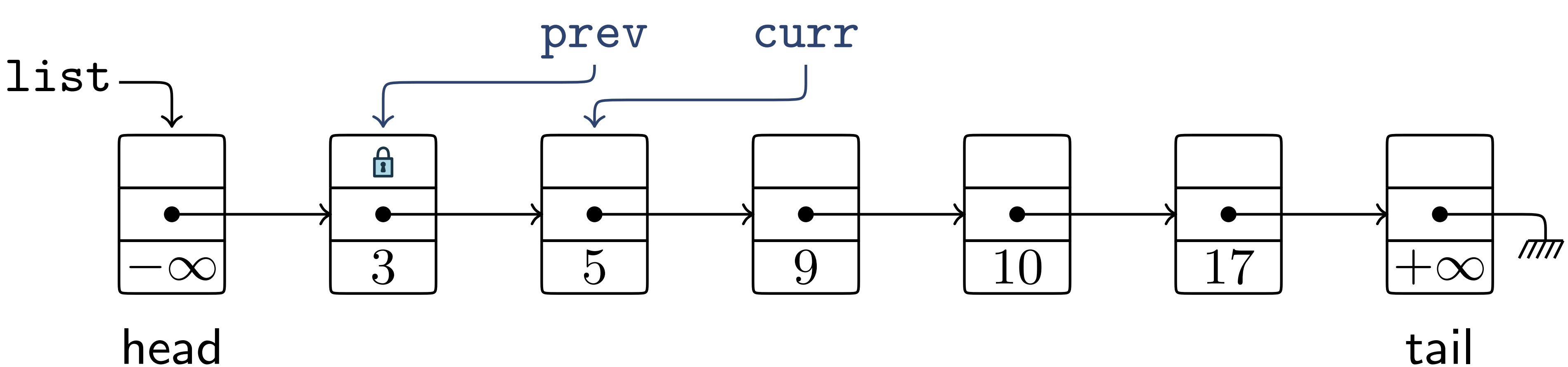
- A concurrent implementation of sets



$search(8)$

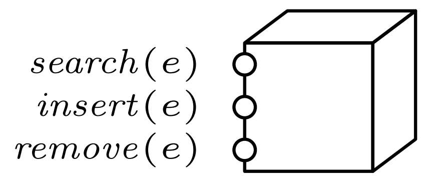
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

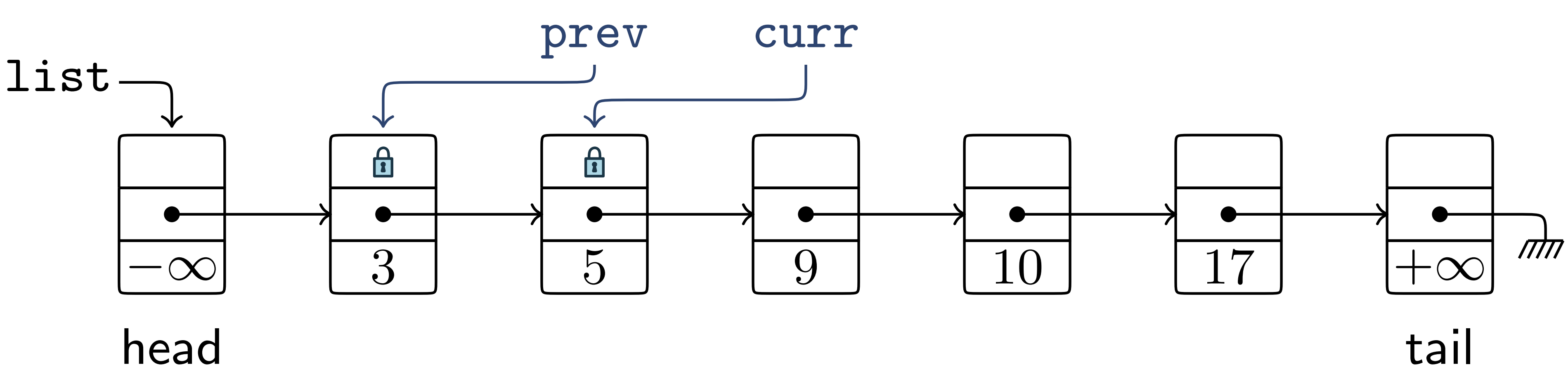
- A concurrent implementation of sets



search(8)

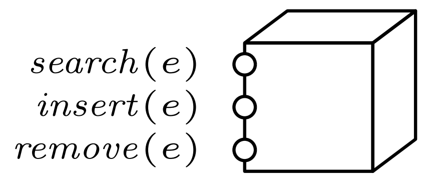
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

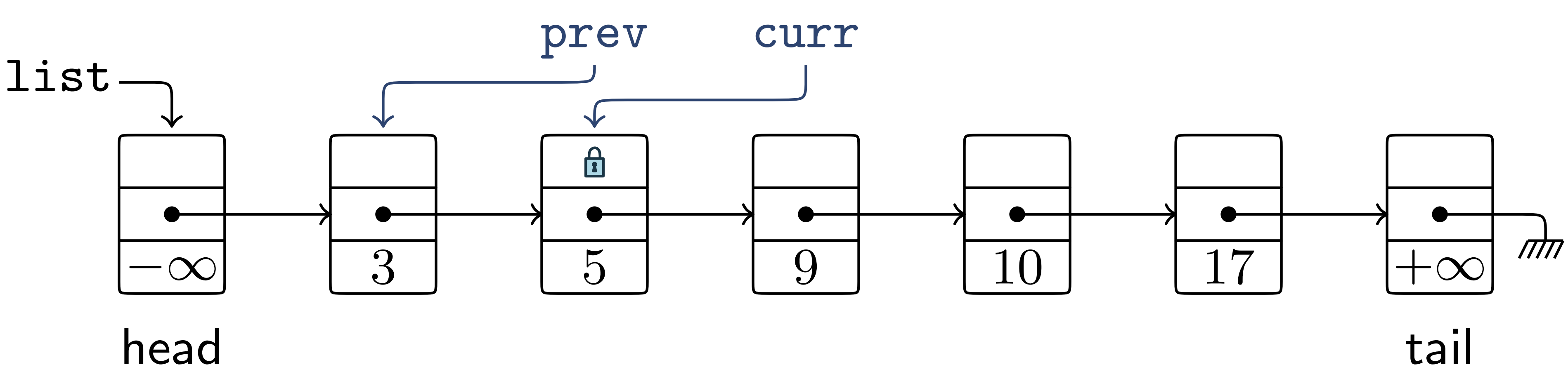
- A concurrent implementation of sets



$search(8)$

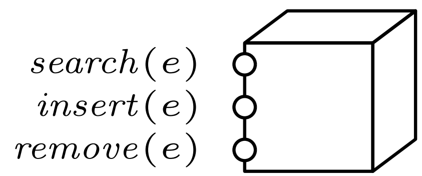
```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



Concurrent Lock-coupling Lists

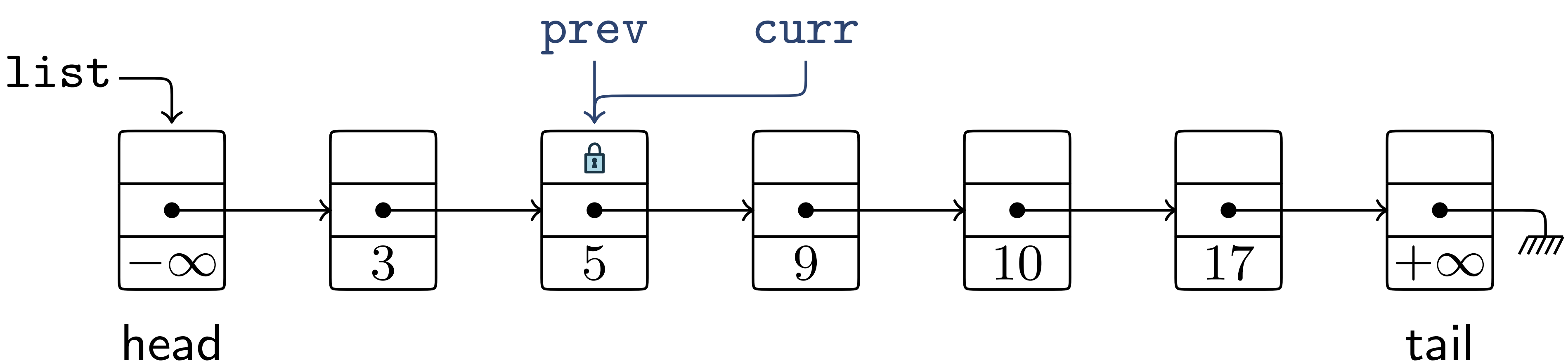
- A concurrent implementation of sets



search(8)

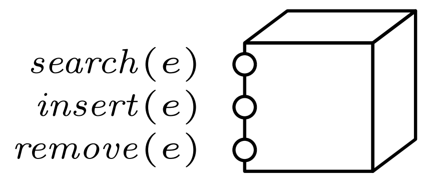
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

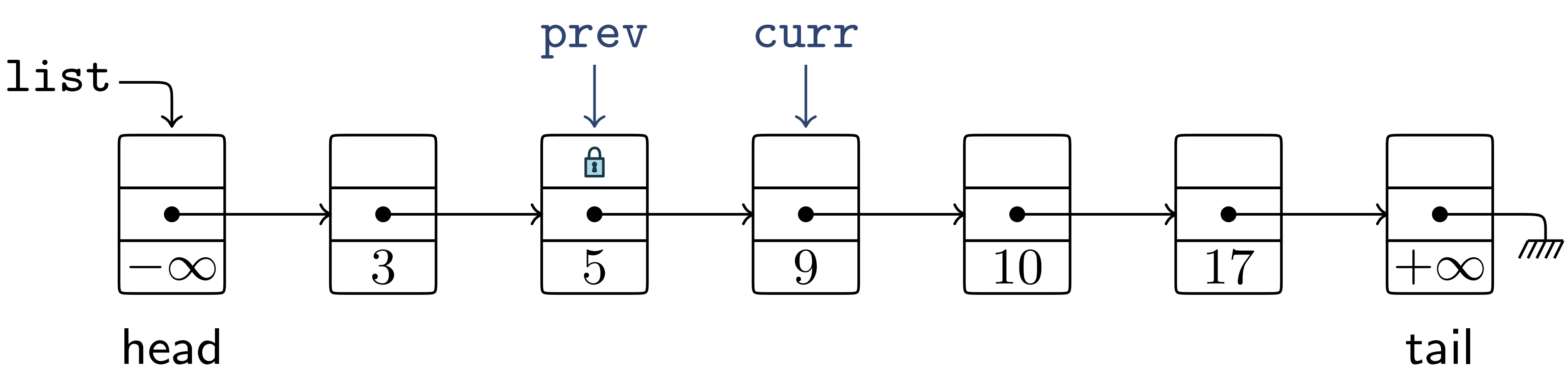
- A concurrent implementation of sets



search(8)

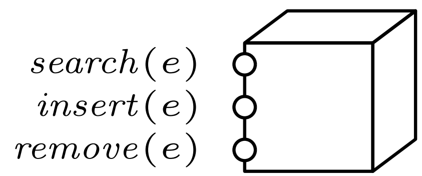
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

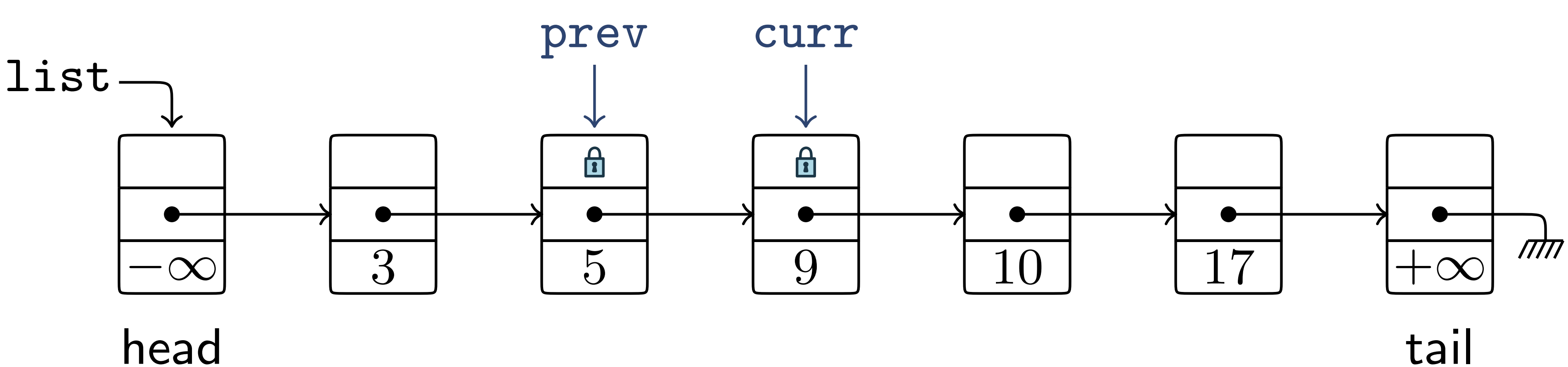
- A concurrent implementation of sets



$search(8)$

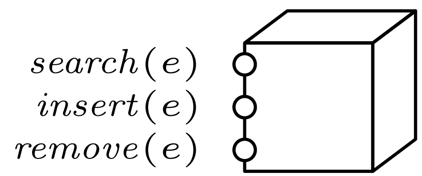
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

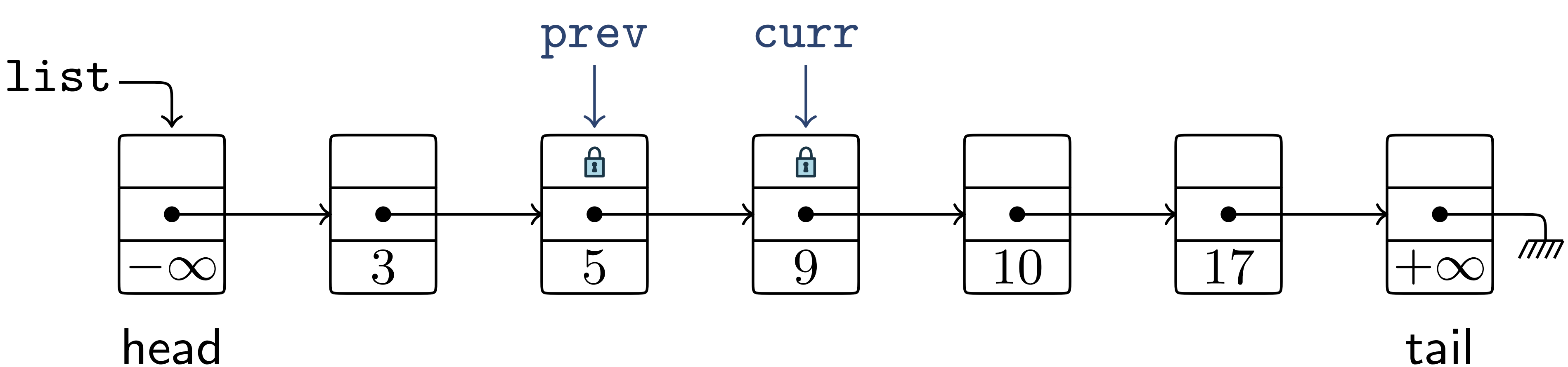
- A concurrent implementation of sets



$insert(8)$

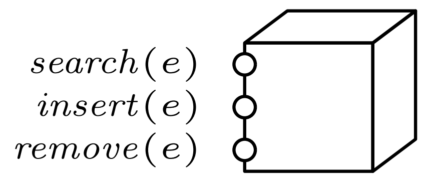
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

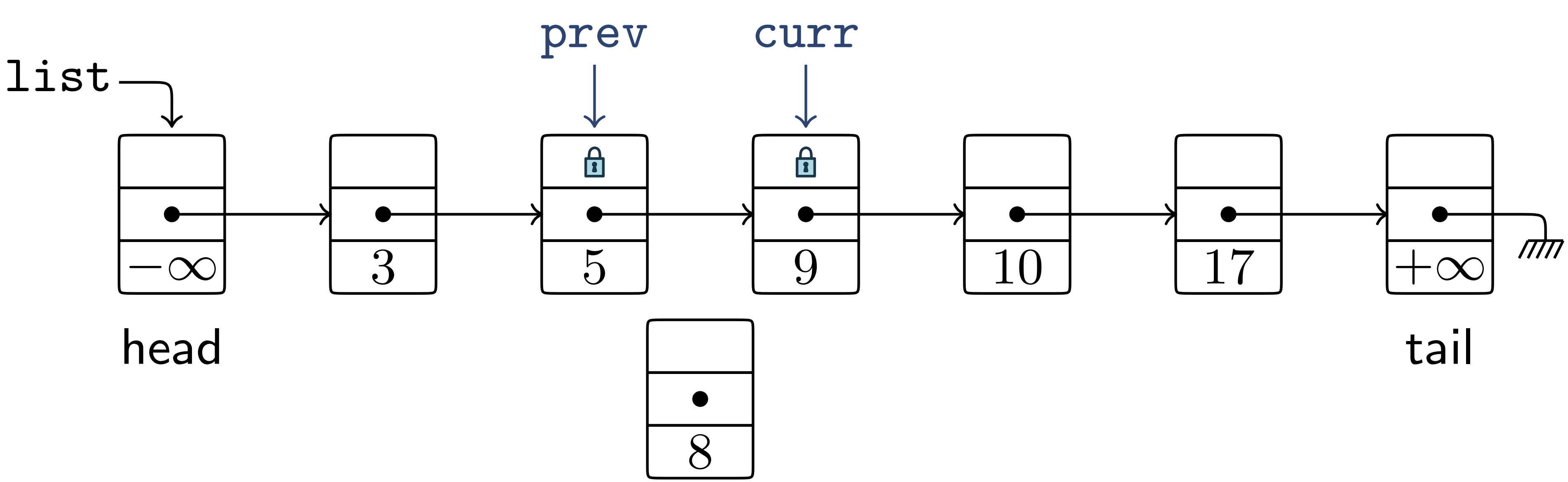
- A concurrent implementation of sets



$insert(8)$

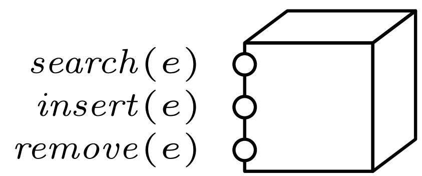
```
class Node {
     Value val;
     Node next;
     Lock lock;
}
```

```
class List {
     Node list;
}
```



Concurrent Lock-coupling Lists

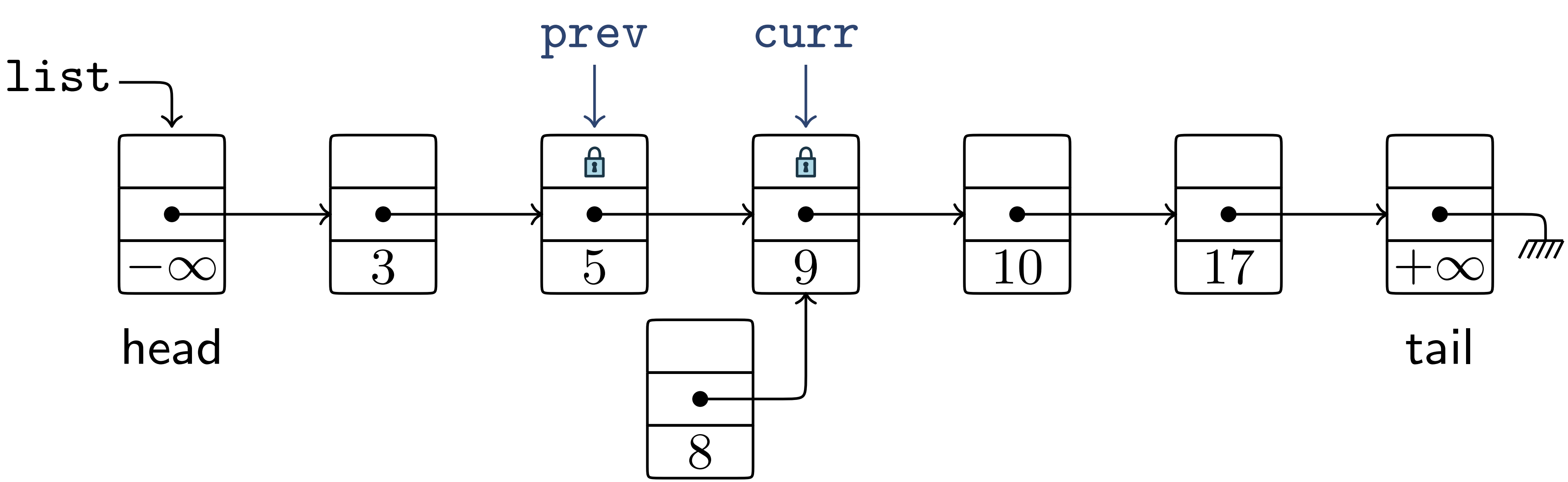
- A concurrent implementation of sets



insert(8)

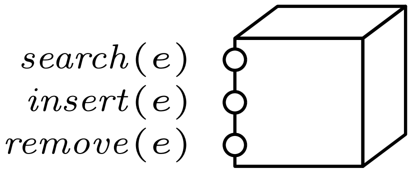
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

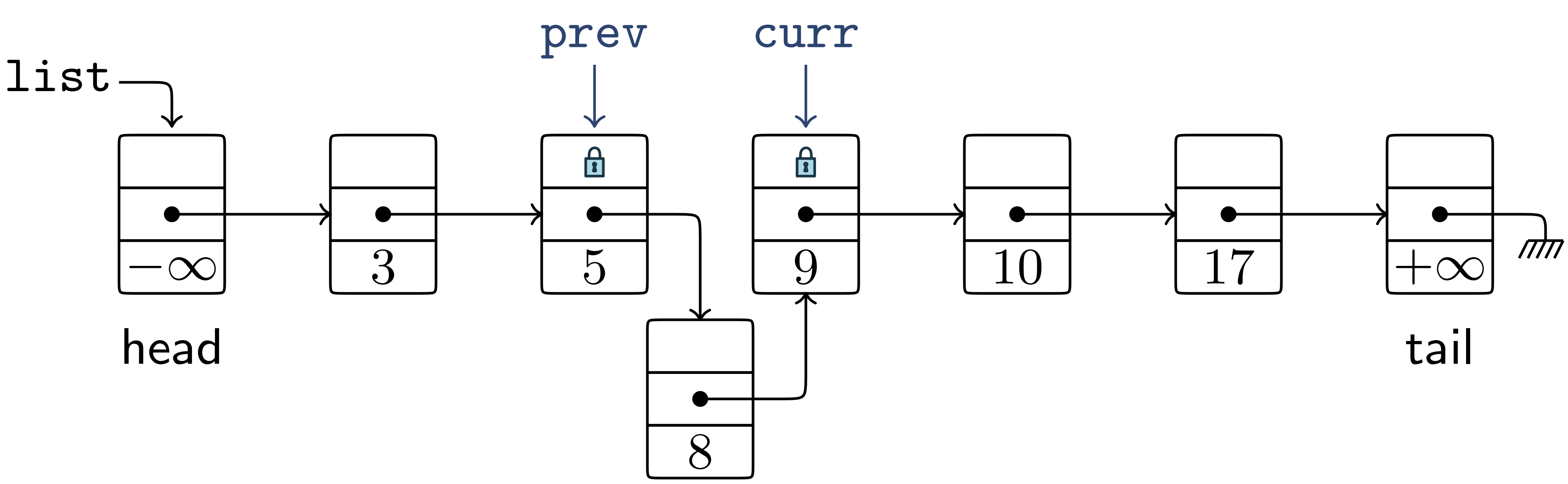
- A concurrent implementation of sets



insert(8)

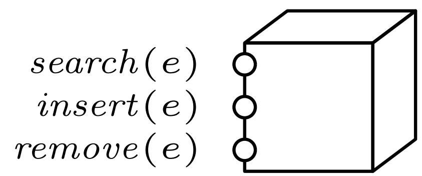
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



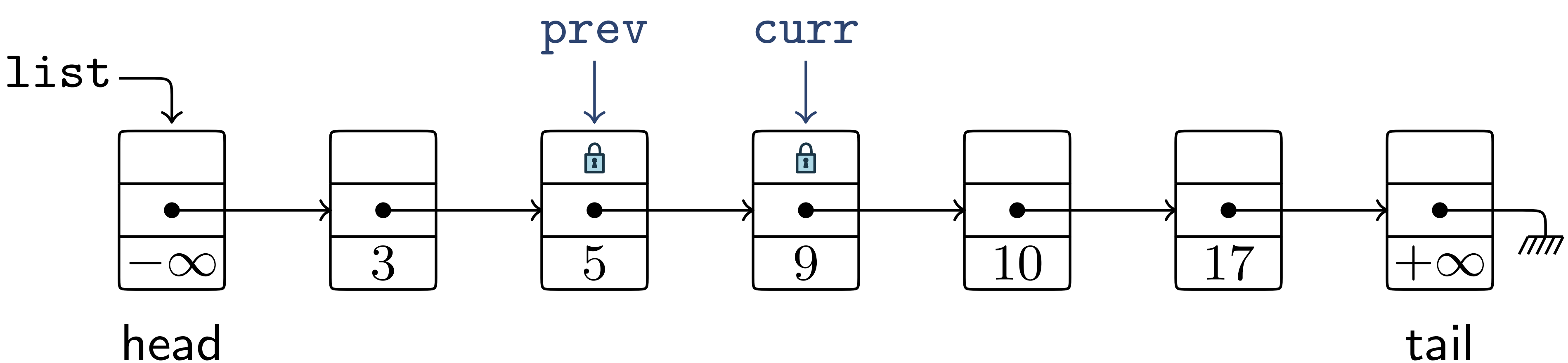
Concurrent Lock-coupling Lists

- A concurrent implementation of sets



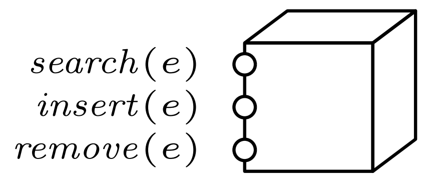

```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

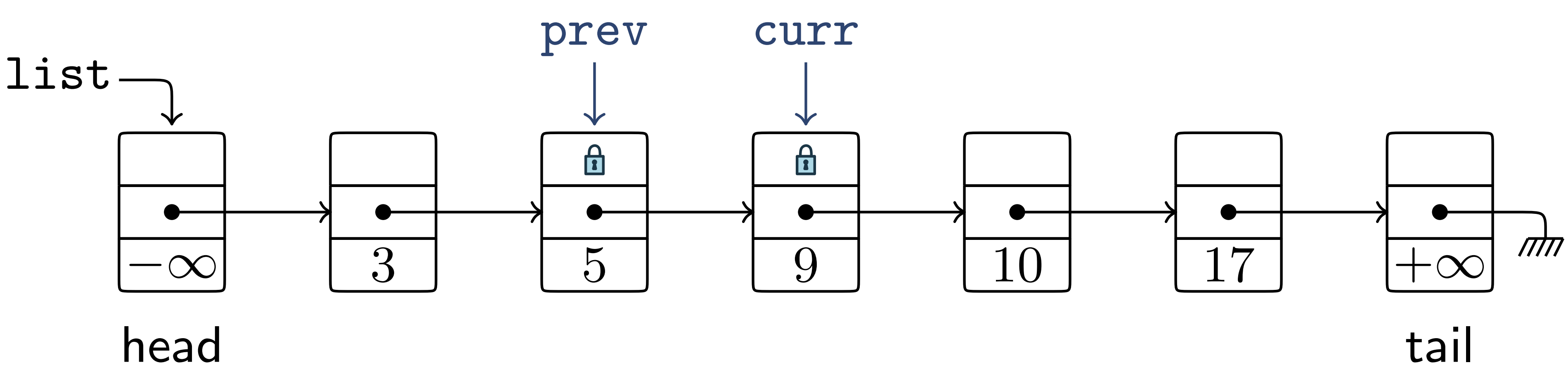
- A concurrent implementation of sets



$remove(9)$

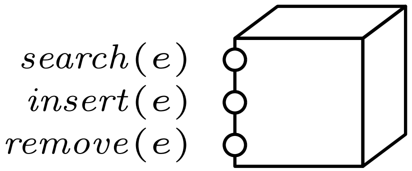
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

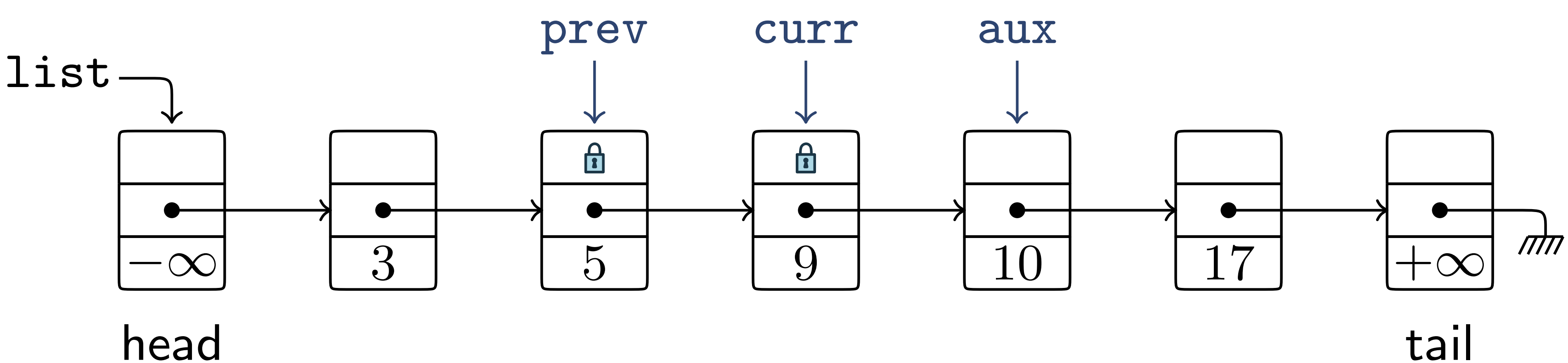
- A concurrent implementation of sets



$remove(9)$

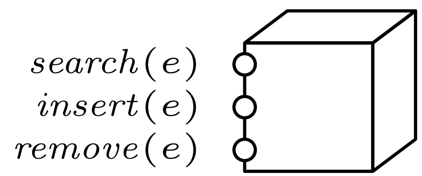
```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Concurrent Lock-coupling Lists

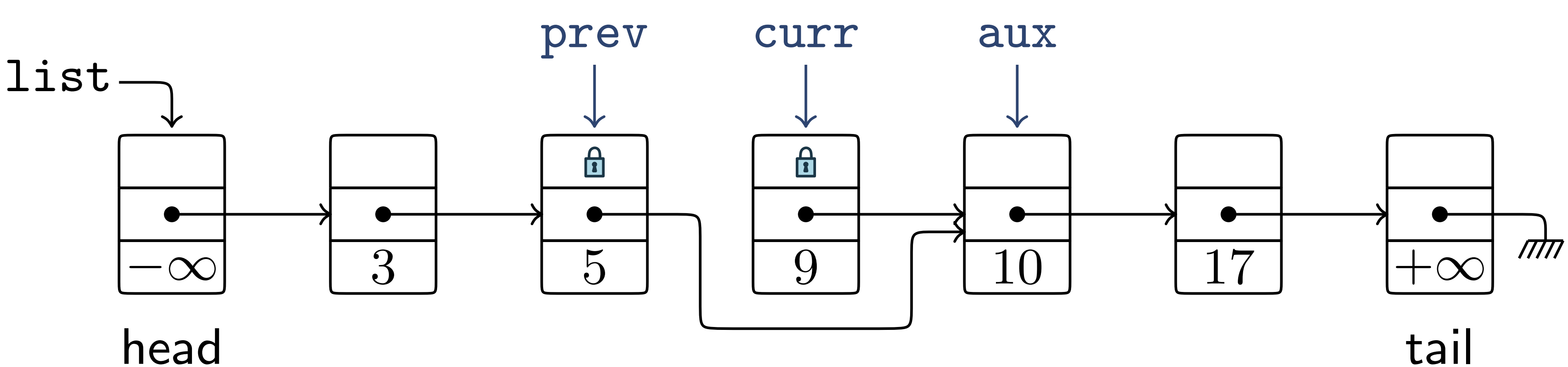
- A concurrent implementation of sets



$remove(9)$

```
class Node {
      Value val;
      Node next;
      Lock lock;
}
```

```
class List {
      Node list;
}
```



Verification Diagram for "Last Terminates"

Verification Diagram for "Last Terminates"

$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"

Concurrent execution of N "most general clients"

$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"

Concurrent execution of N "most general clients"

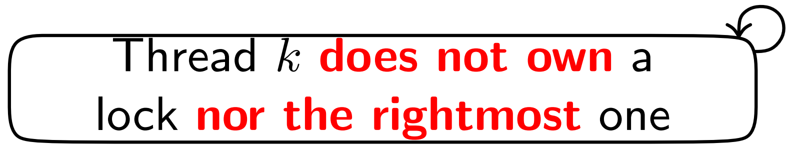
$$\mathcal{S}[N] \models \psi^{(k)}$$

Thread k holding the rightmost lock, terminates

Verification Diagram for "Last Terminates"

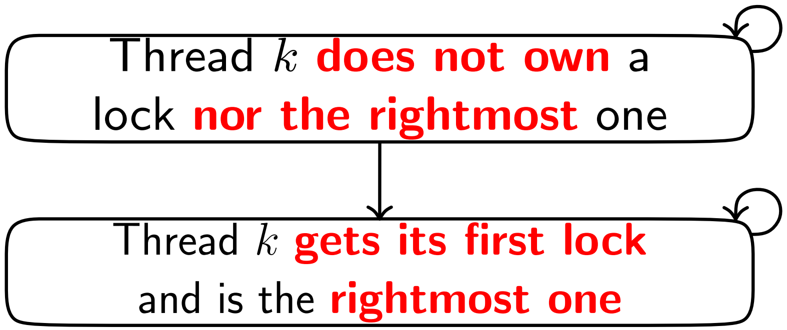
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



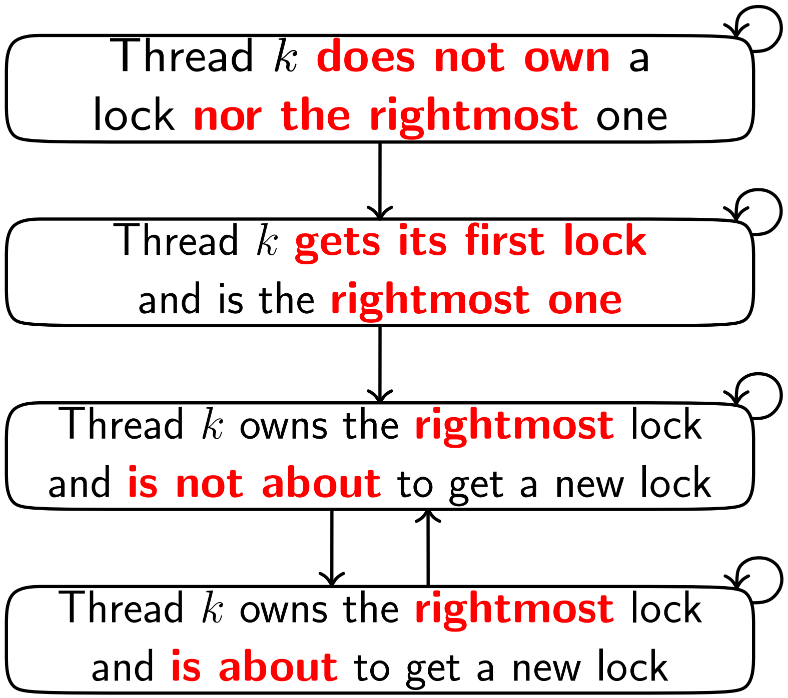
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



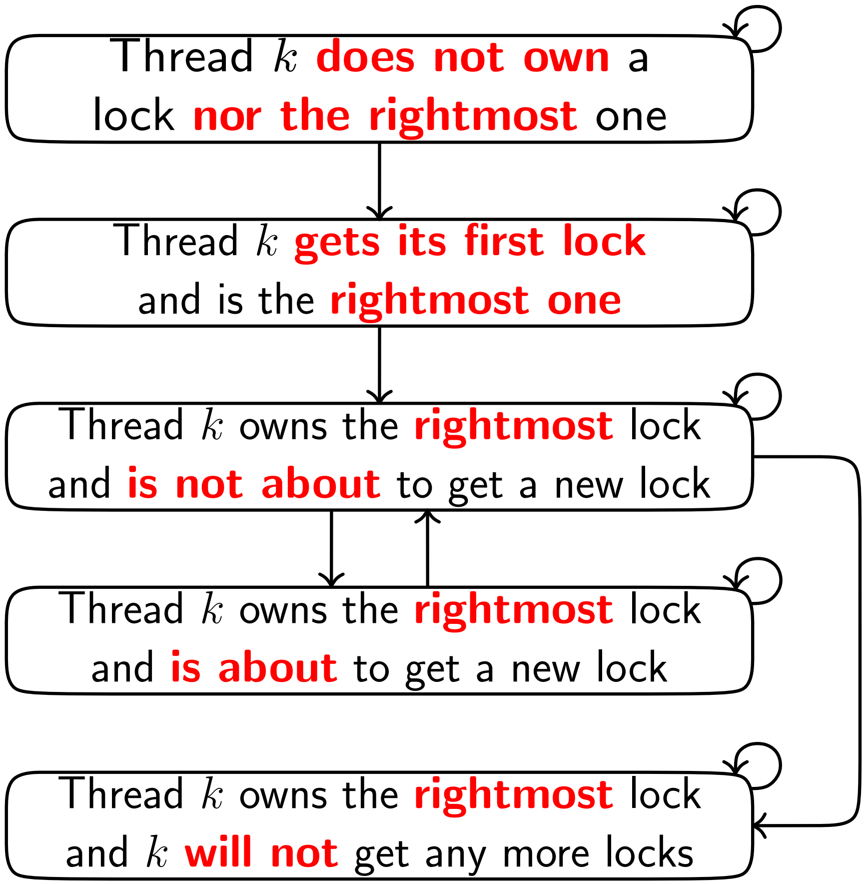
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



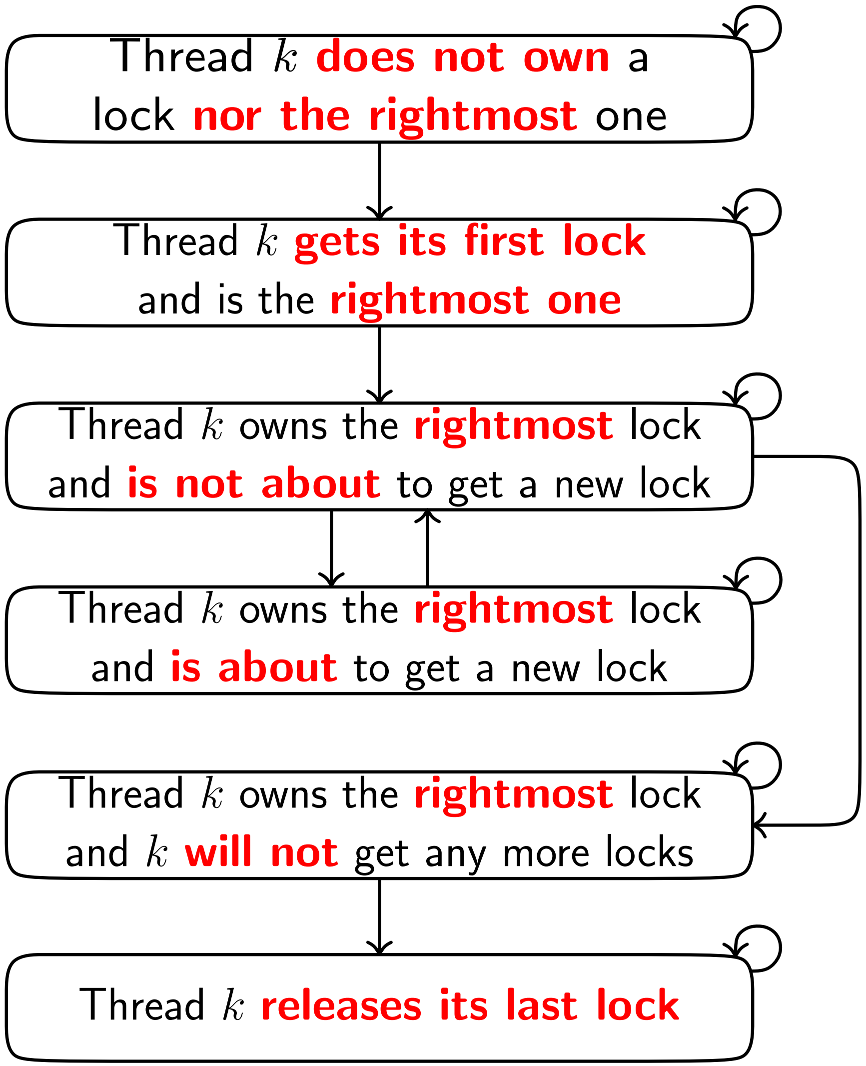
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



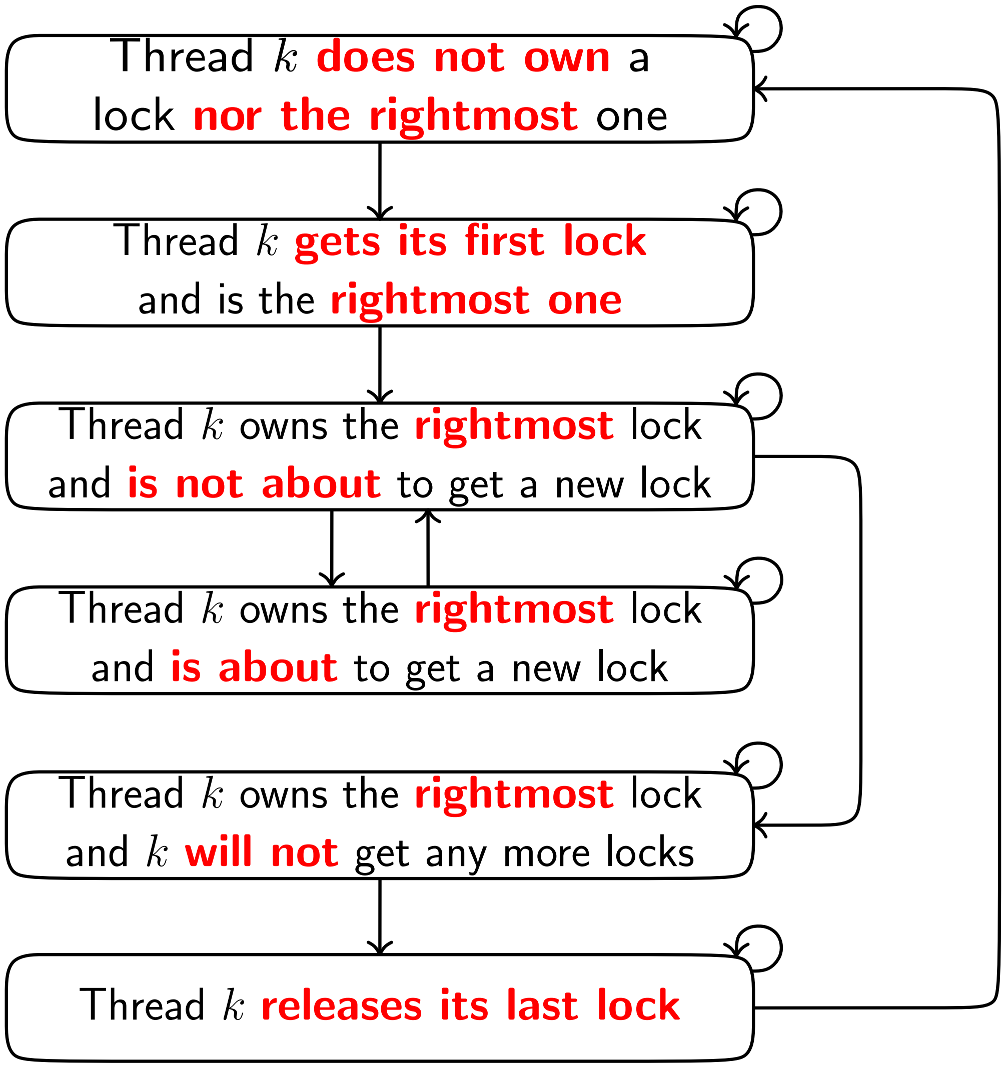
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



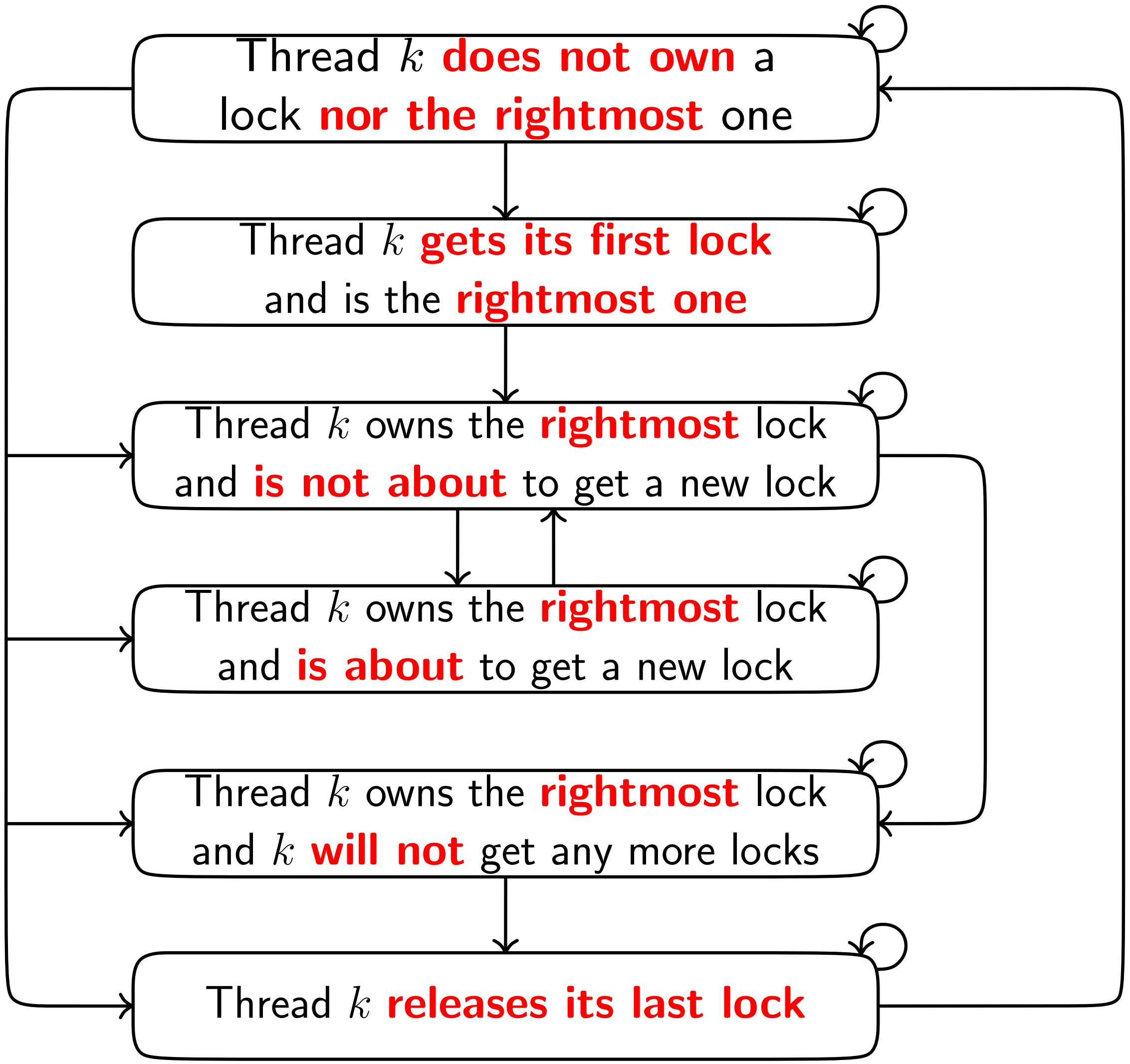
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



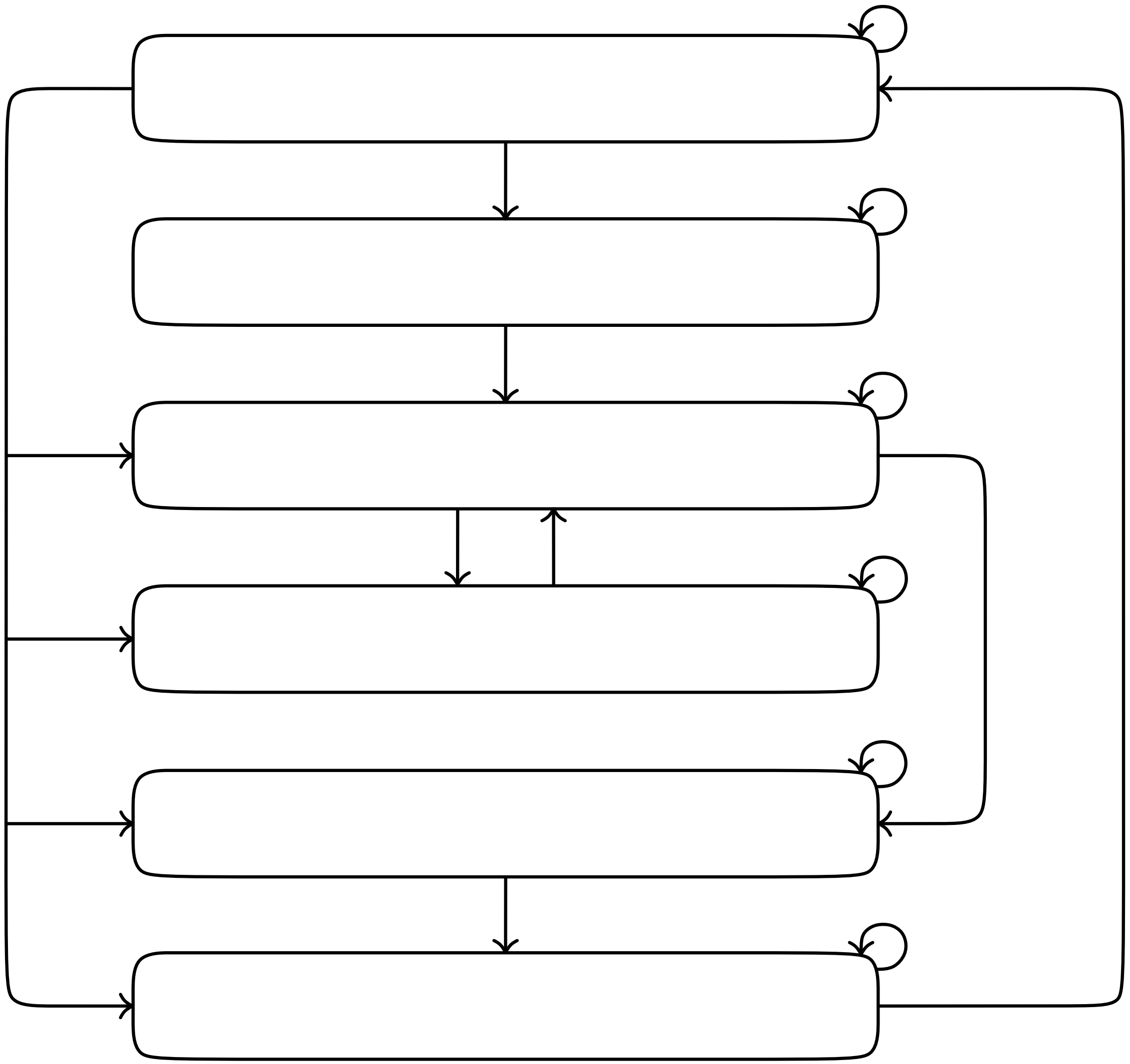
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



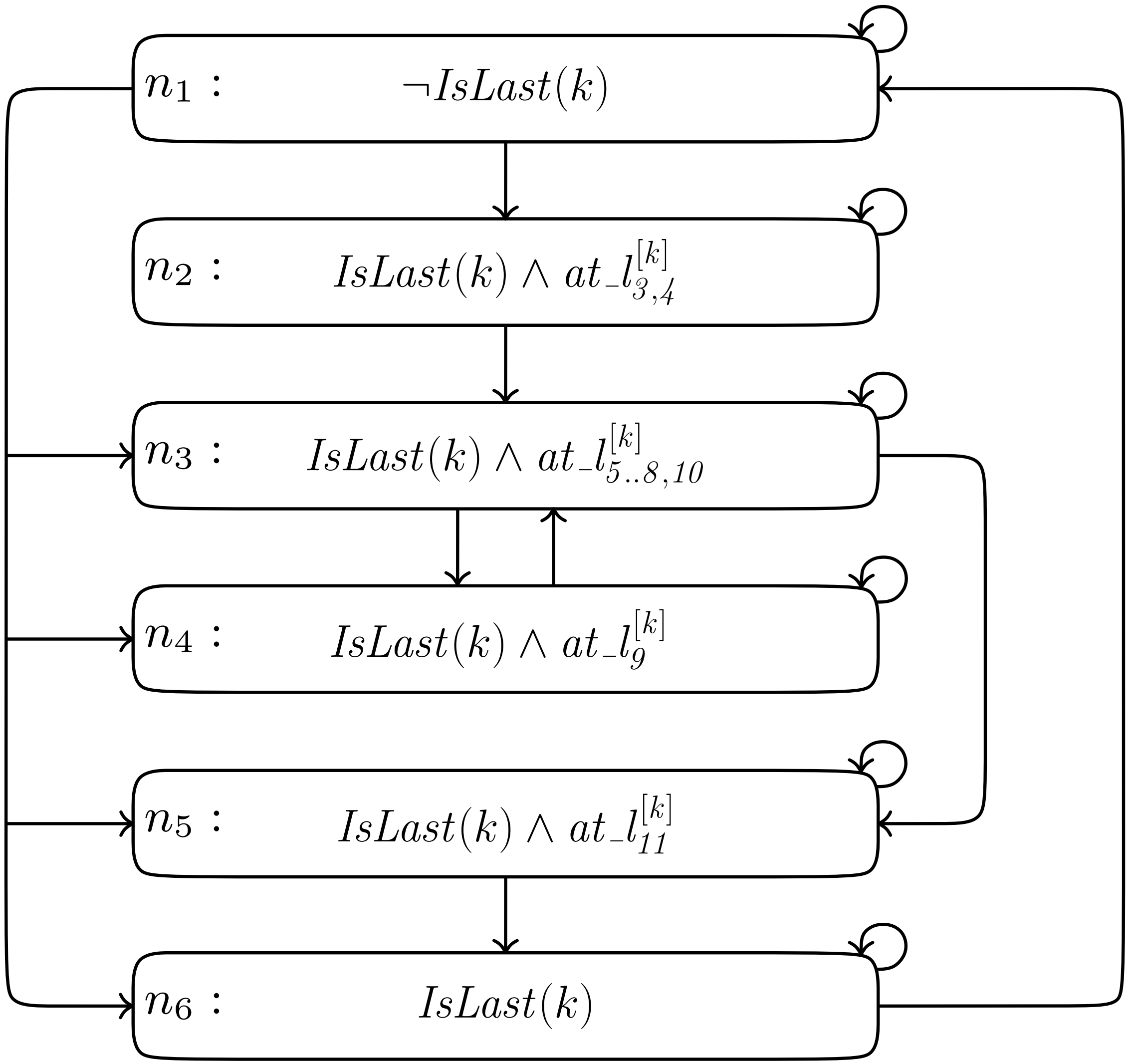
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



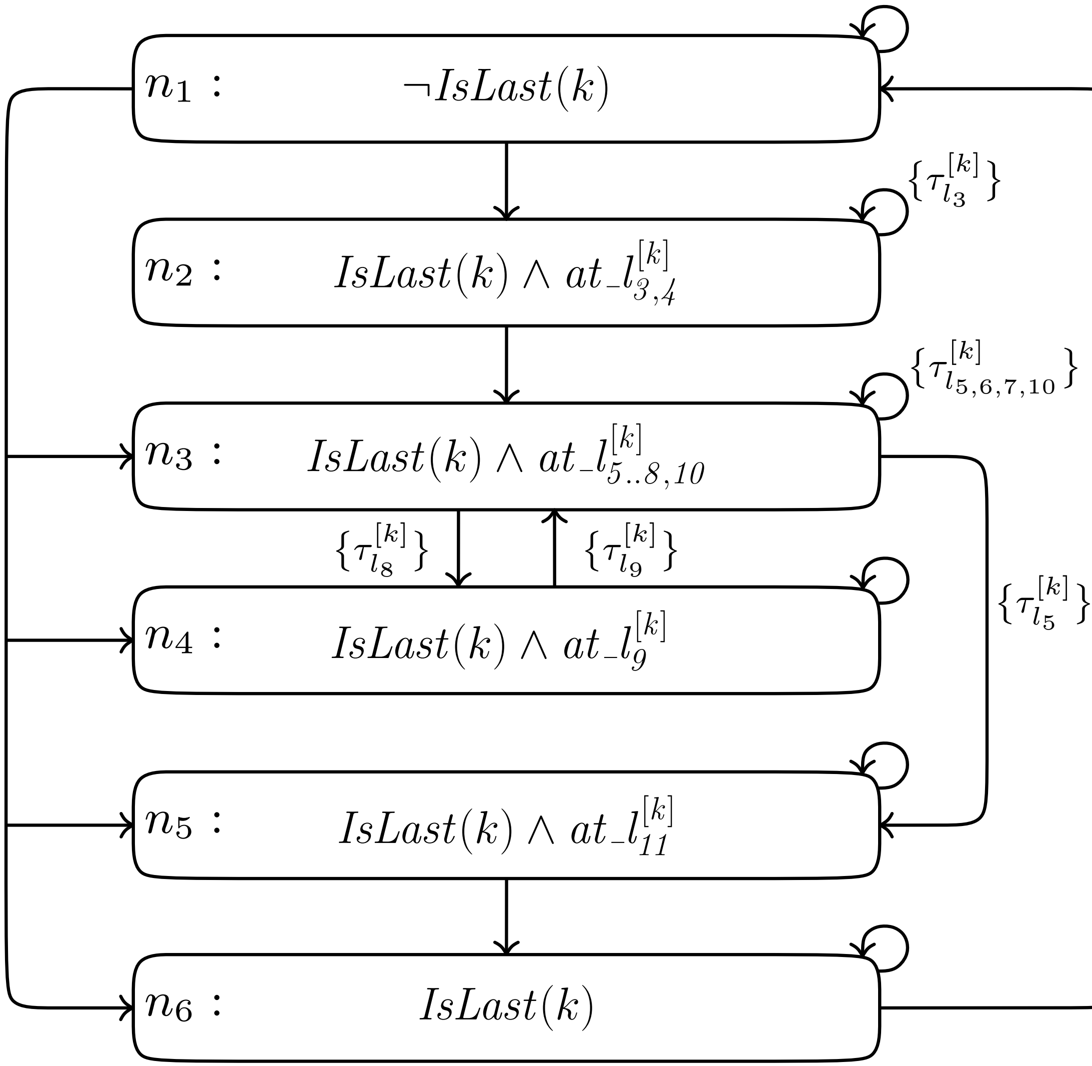
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



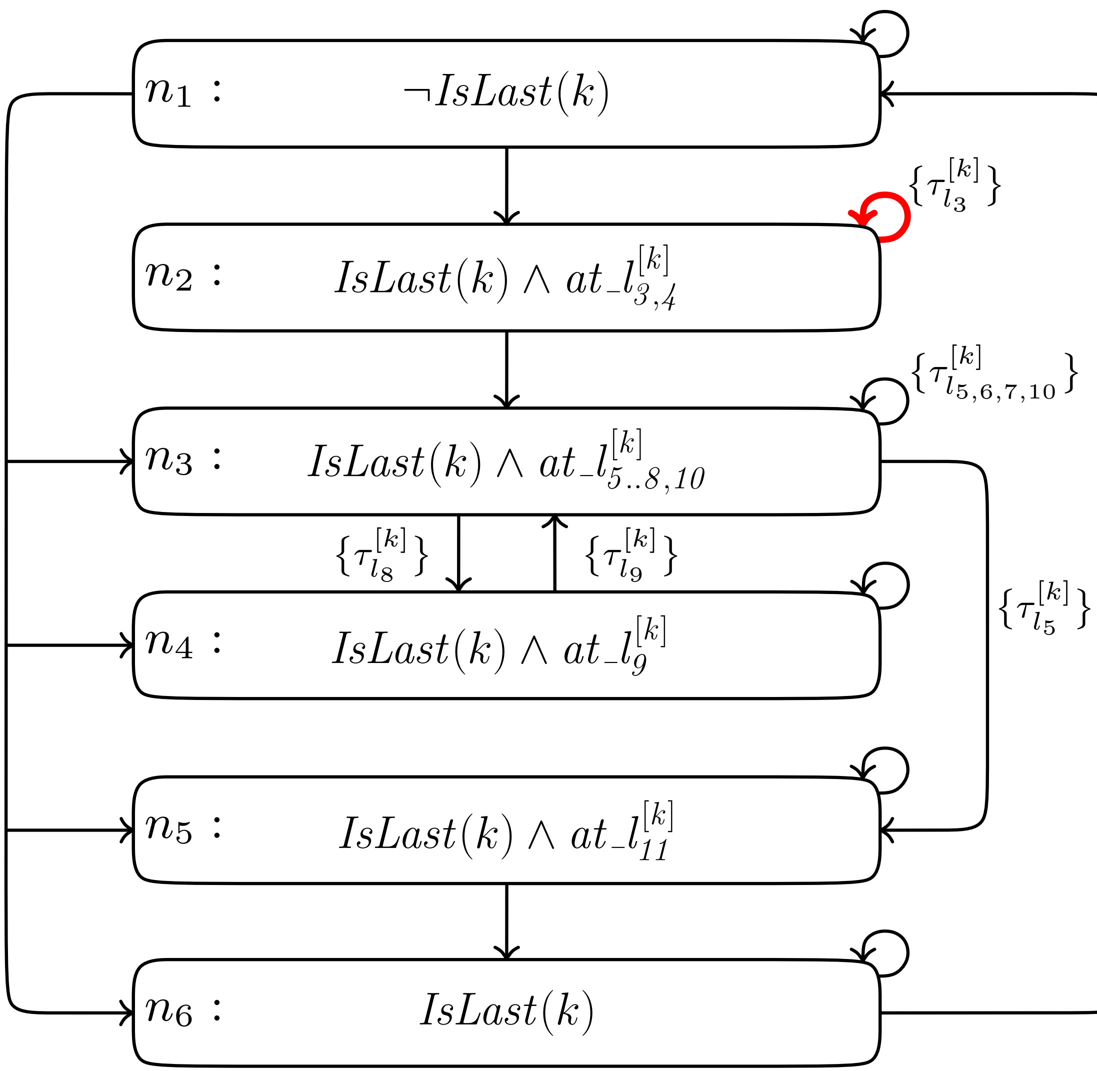
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"



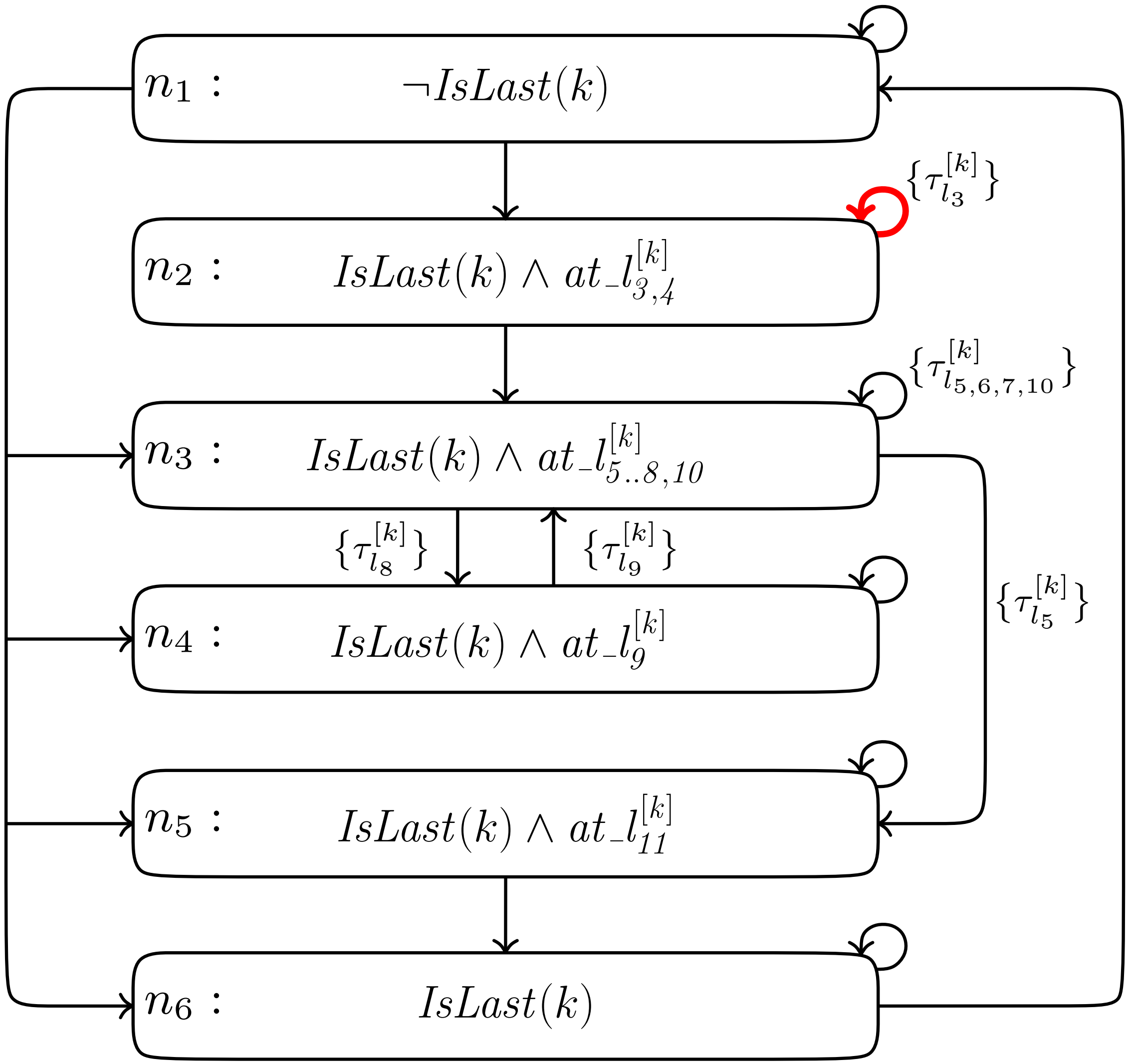
$$\mathcal{S}[N] \models \psi^{(k)}$$

Verification Diagram for "Last Terminates"

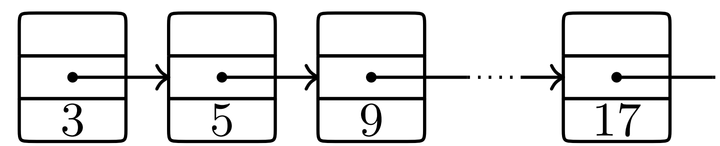


$$\mathcal{S}[N] \models \psi^{(k)}$$

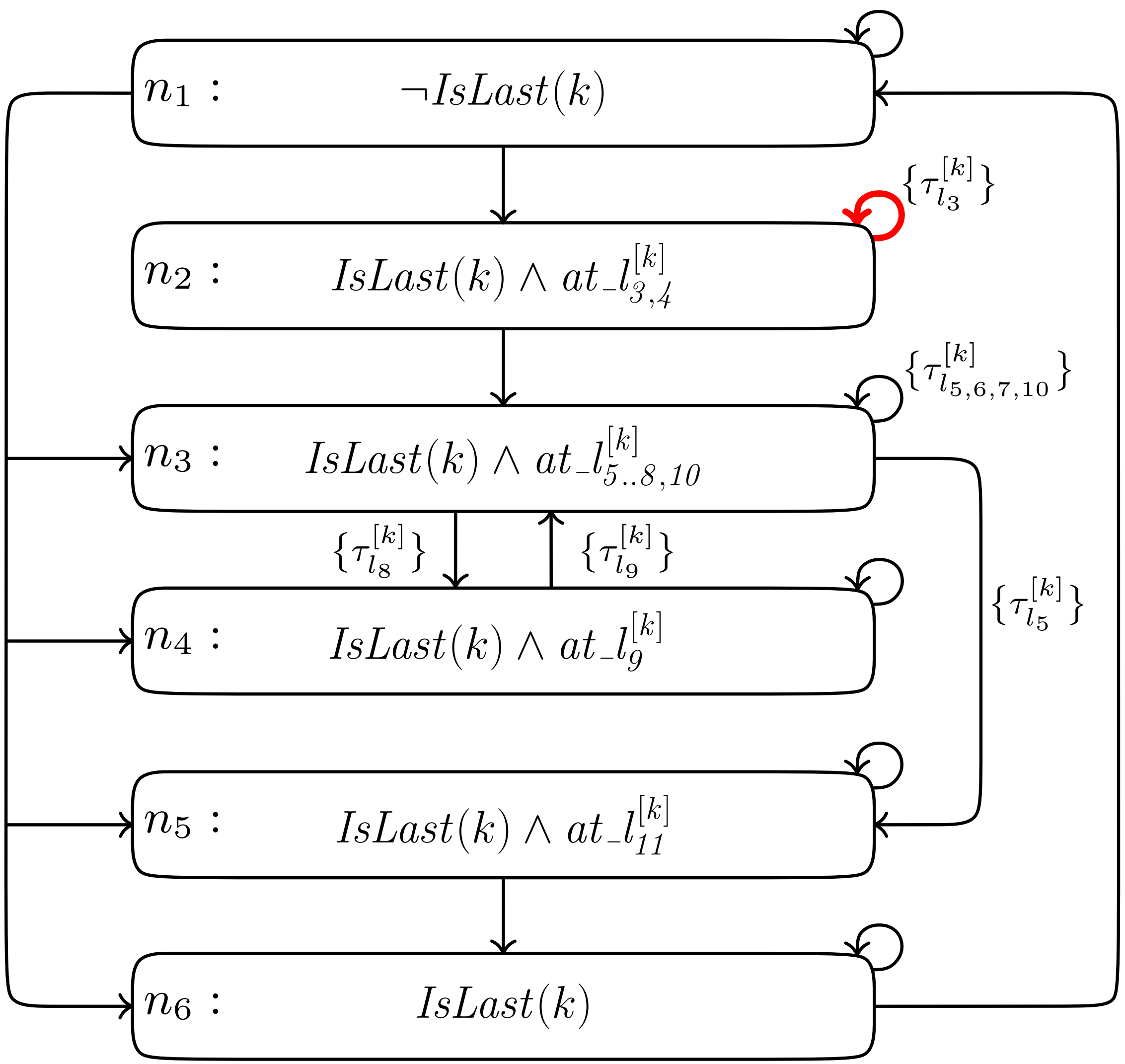
Verification Diagram for "Last Terminates"



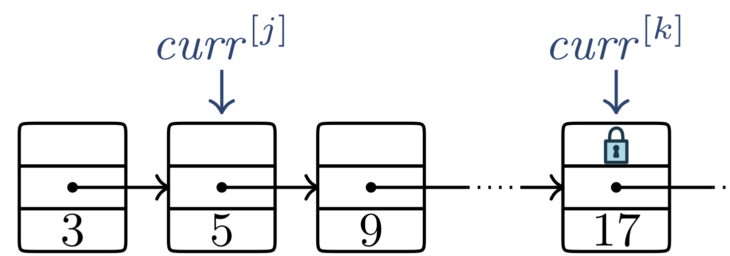
$$\mathcal{S}[N] \vDash \psi^{(k)}$$



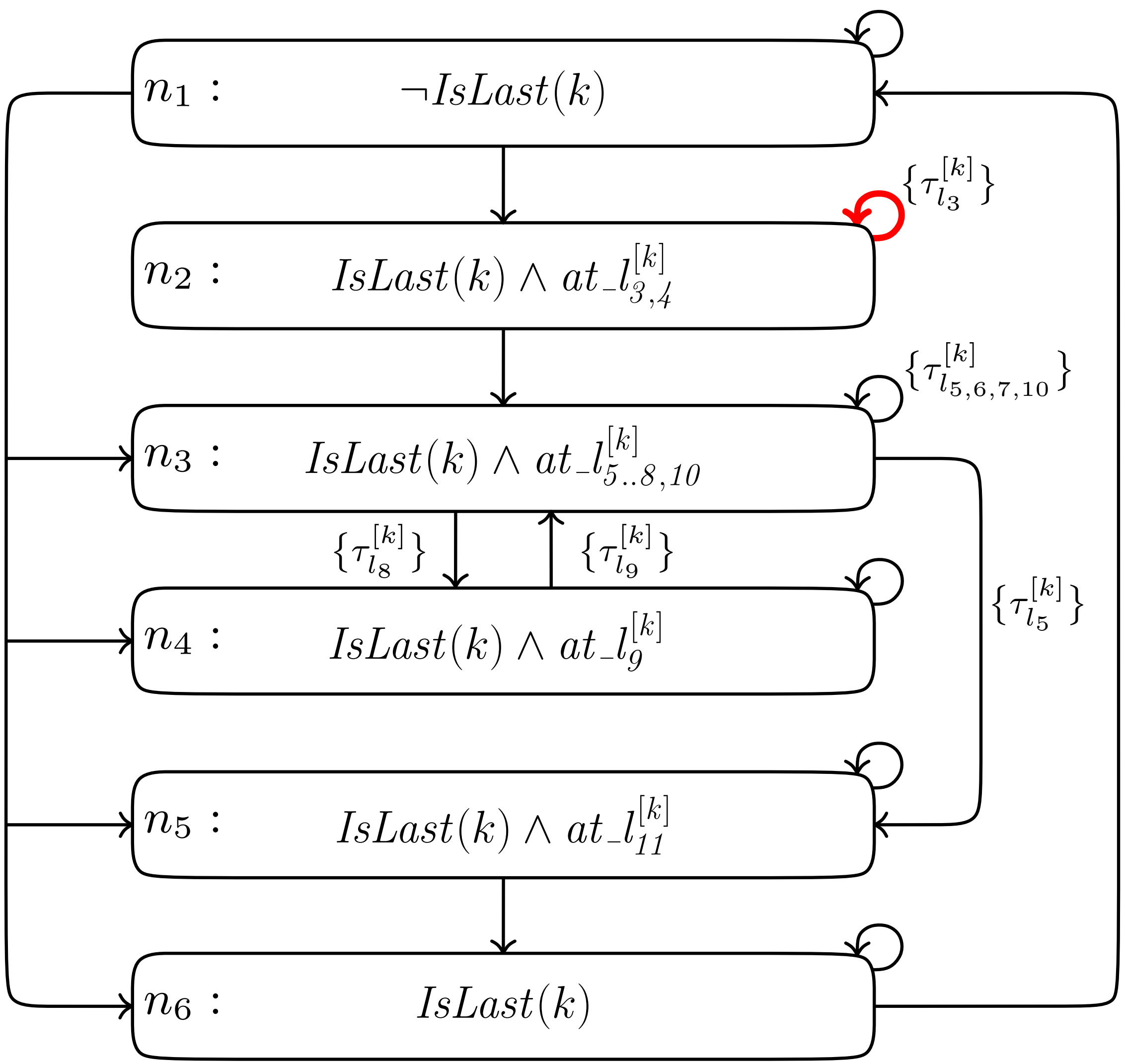
Verification Diagram for "Last Terminates"



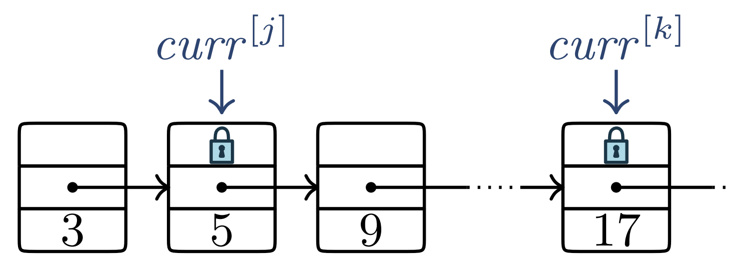
$$\mathcal{S}[N] \vDash \psi^{(k)}$$



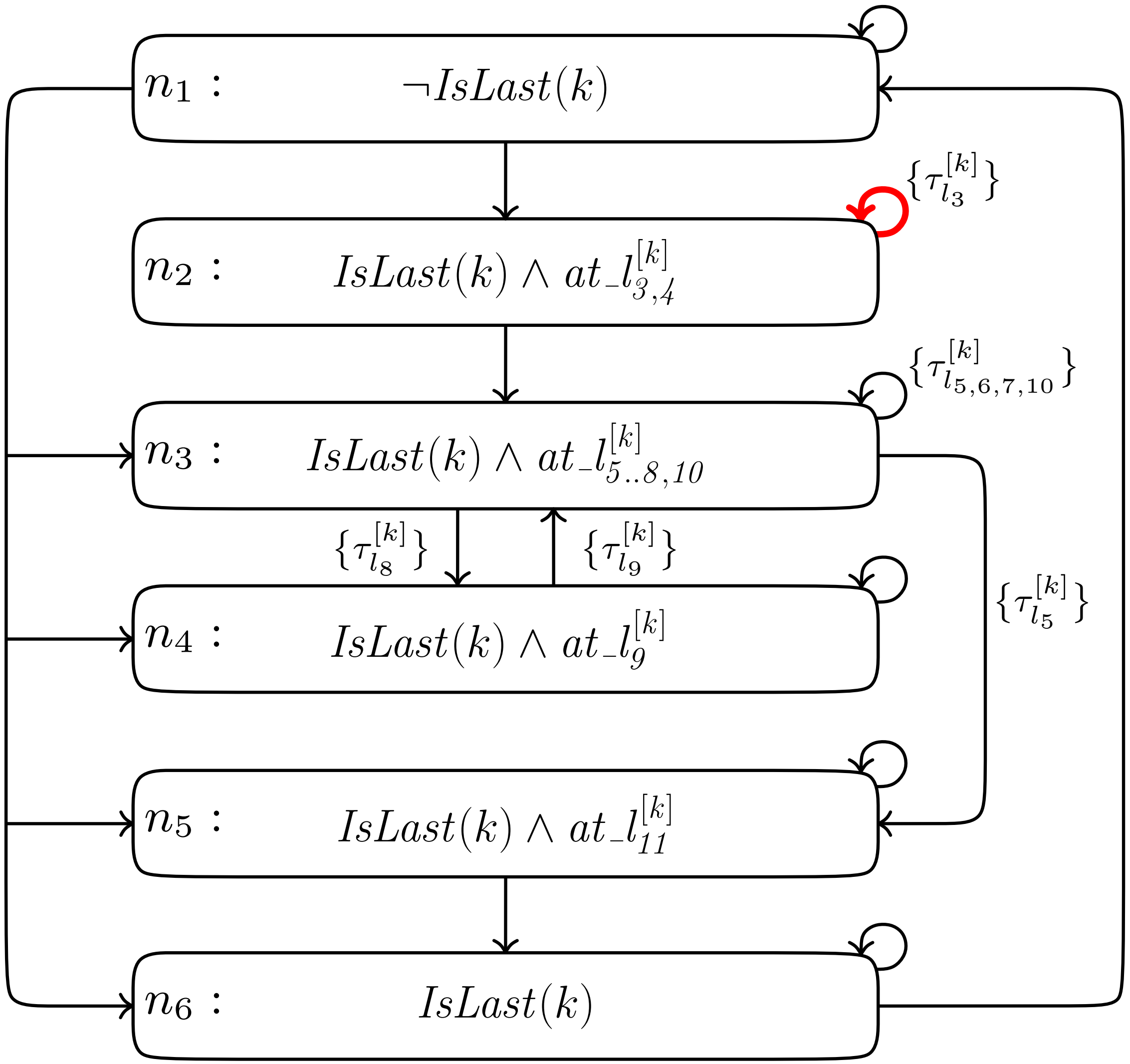
Verification Diagram for "Last Terminates"



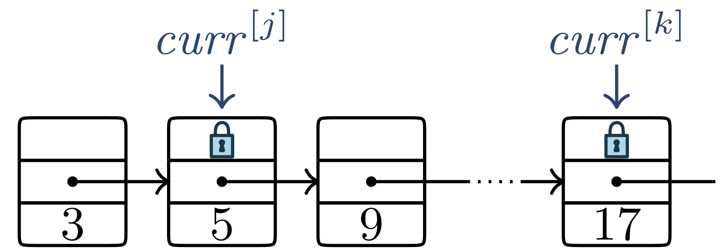
$$\mathcal{S}[N] \vDash \psi^{(k)}$$



Verification Diagram for "Last Terminates"

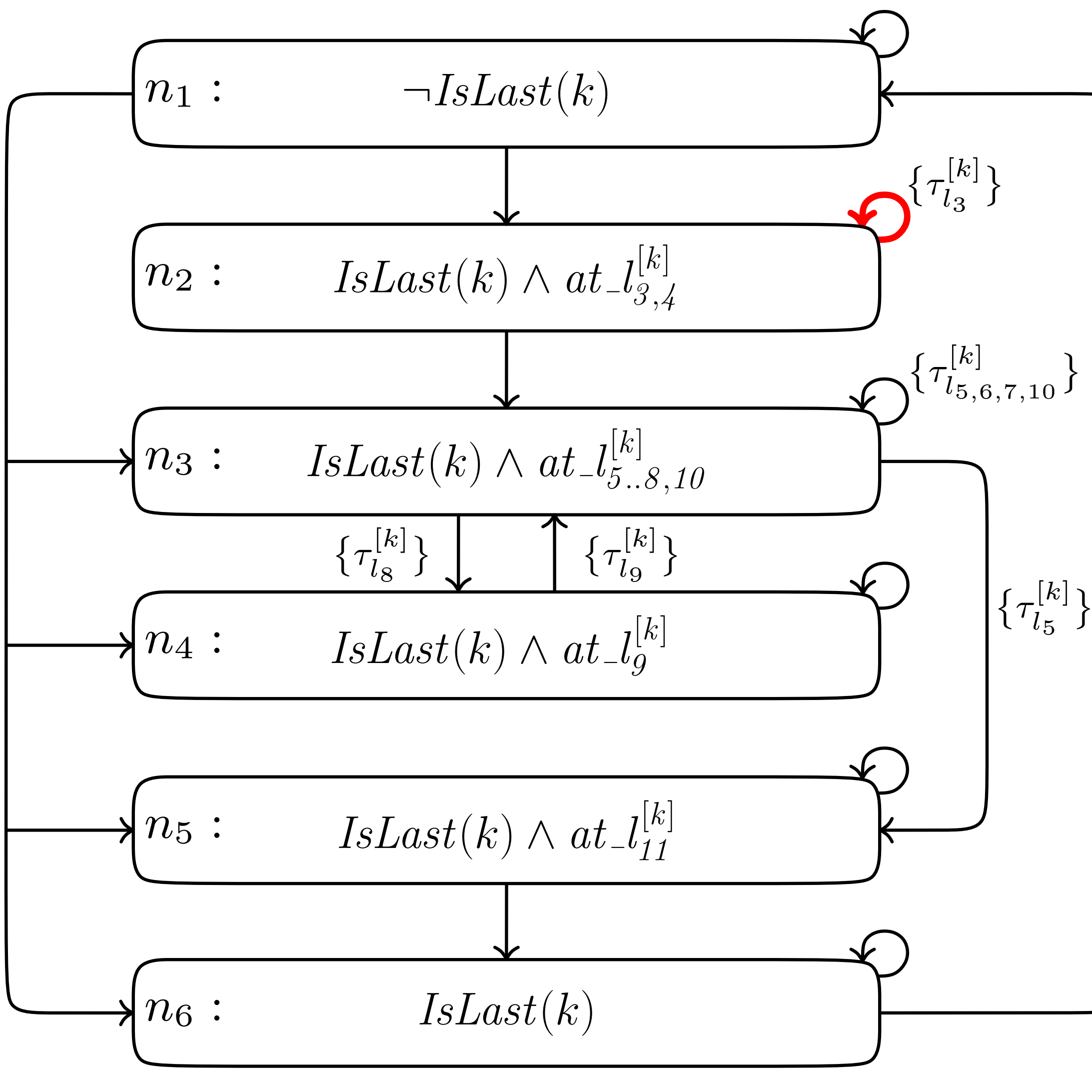


$$\mathcal{S}[N] \vDash \psi^{(k)}$$

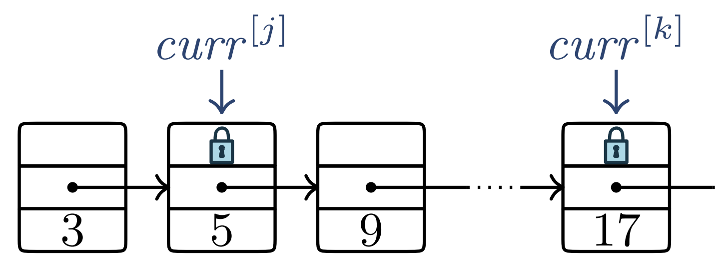


$$\begin{pmatrix} IsLast(k) \wedge \ j \neq k \wedge \\ at\_l^{[k]}_{3,4} \wedge at\_l^{[j]}_{9} \wedge \\ curr^{[j]}.lockid = \oslash \end{pmatrix} \wedge \ curr^{[j]}.lock(j) \rightarrow \begin{pmatrix} IsLast(k') \wedge at'\_l^{[k']}_{3,4} \wedge \\ at'\_l^{[j']}_{10} \wedge pres(V - curr^{[j]}) \wedge \\ curr'^{[j']}.lockid = j' \end{pmatrix}$$

Verification Diagram for "Last Terminates"



$$\mathcal{S}[N] \models \psi^{(k)}$$

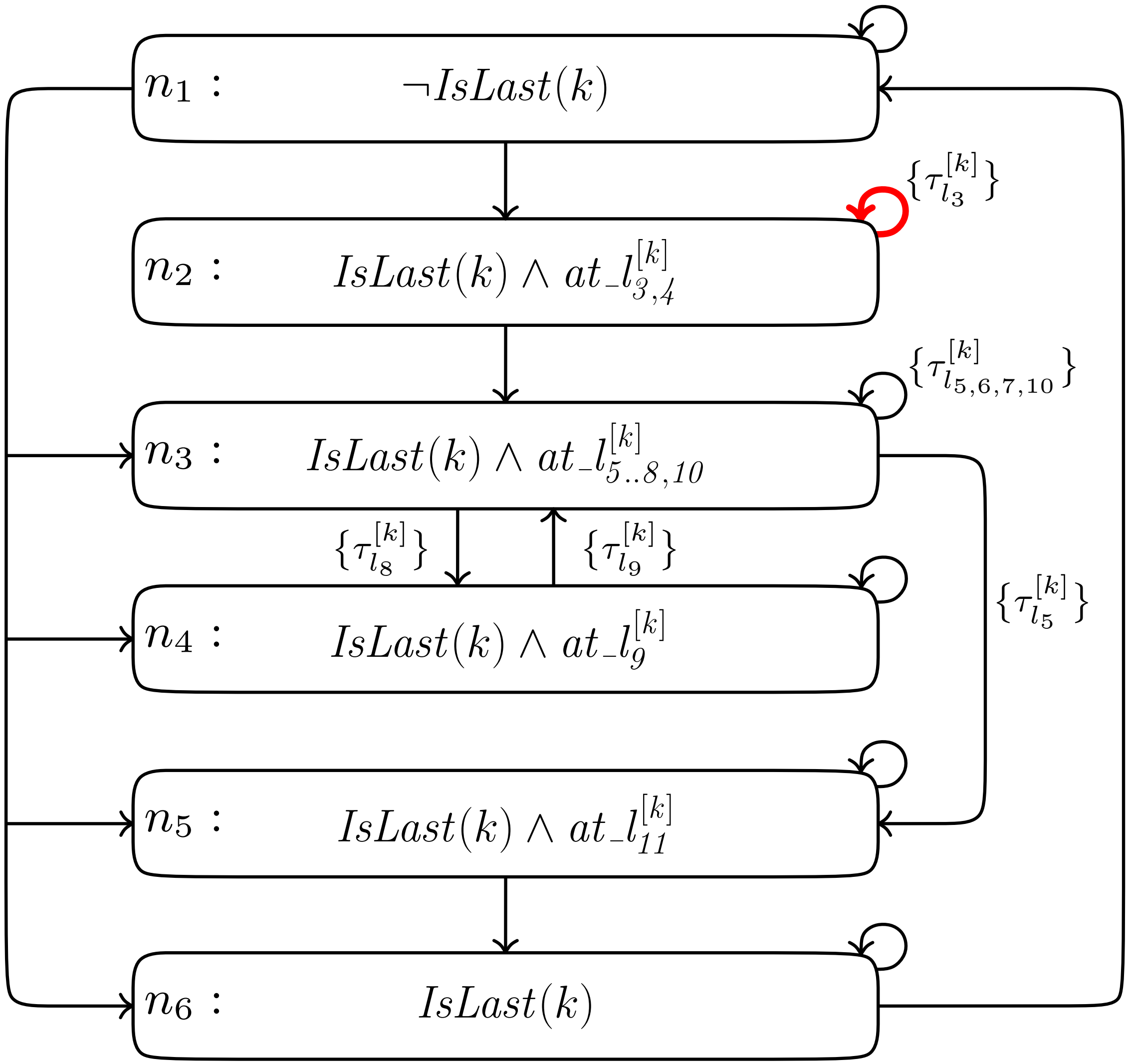


Reason about:

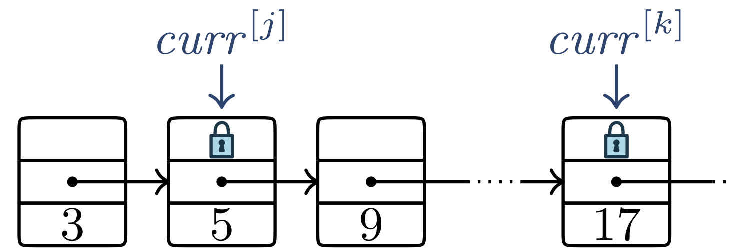
locks

$$\begin{pmatrix} IsLast(k) \wedge \ j \neq k \ \wedge \\ at\_l_{3,4}^{[k]} \wedge at\_l_{9}^{[j]} \wedge \\ curr^{[j]}.lockid = \oslash \end{pmatrix} \wedge curr^{[j]}.lock(j) \rightarrow \begin{pmatrix} IsLast(k') \wedge at'\_l_{3,4}^{[k']} \ \wedge \\ at'\_l_{10}^{[j']} \wedge pres(V - curr^{[j]}) \ \wedge \\ curr'^{[j']}.lockid = j' \end{pmatrix}$$

Verification Diagram for "Last Terminates"



$$\mathcal{S}[N] \vDash \psi^{(k)}$$

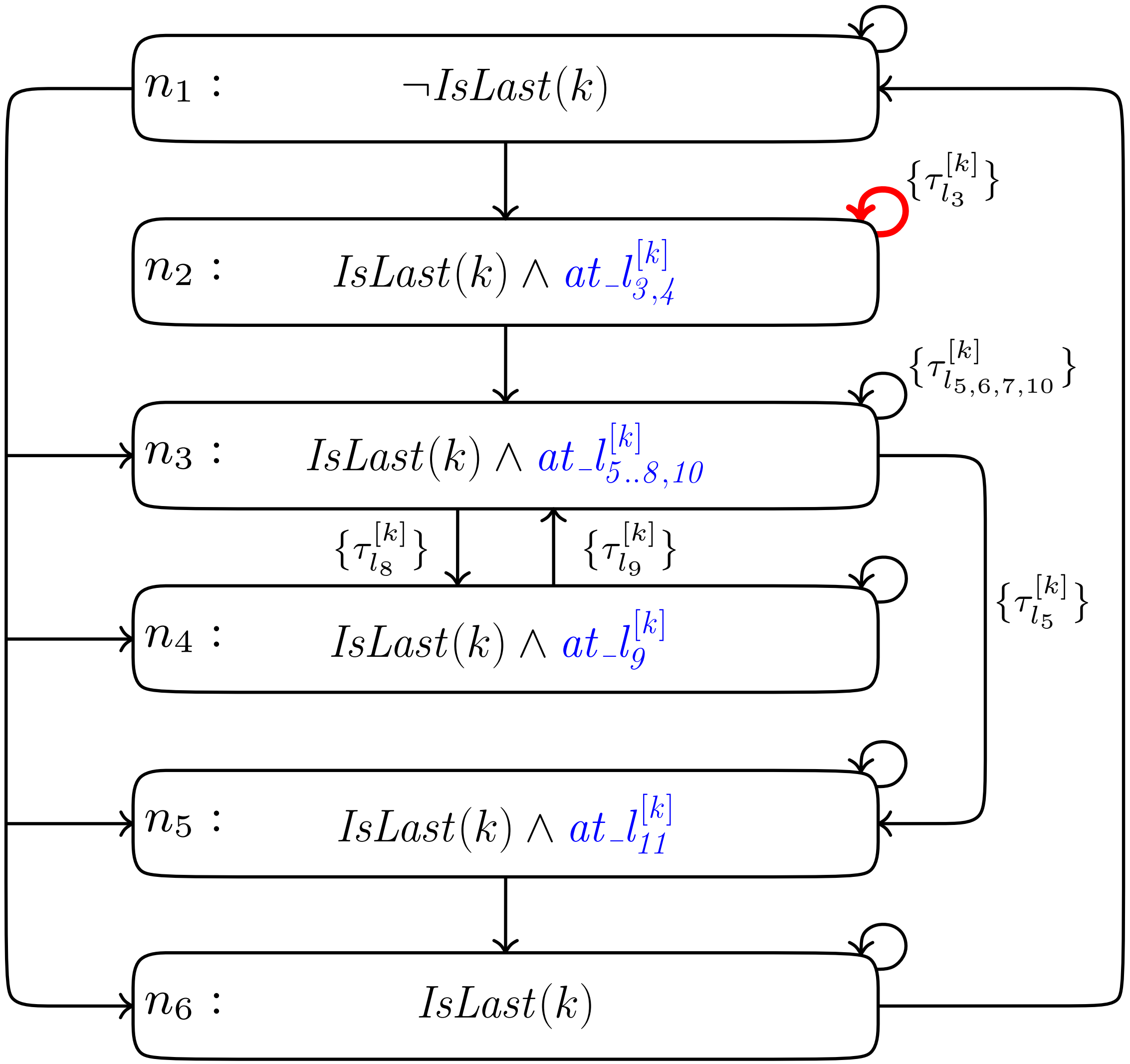


Reason about:

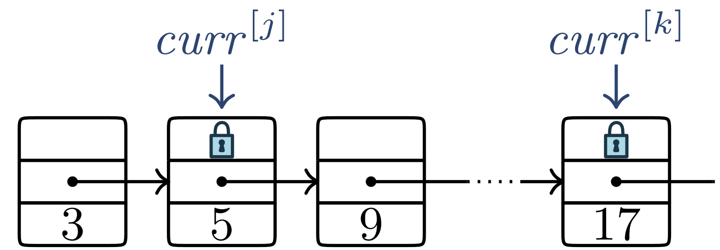
thread identifiers

$$\begin{pmatrix} IsLast(k) \wedge j \neq k \wedge \\ at\_l_{3,4}^{[k]} \wedge at\_l_9^{[j]} \wedge \\ curr^{[j]}.lockid = \emptyset \end{pmatrix} \wedge \; curr^{[j]}.lock(j) \rightarrow \begin{pmatrix} IsLast(k') \wedge at'\_l_{3,4}^{[k']} \wedge \\ at'\_l_{10}^{[j']} \wedge pres(V - curr^{[j]}) \wedge \\ curr'^{[j']}.lockid = j' \end{pmatrix}$$

Verification Diagram for "Last Terminates"



$$\mathcal{S}[N] \vDash \psi^{(k)}$$

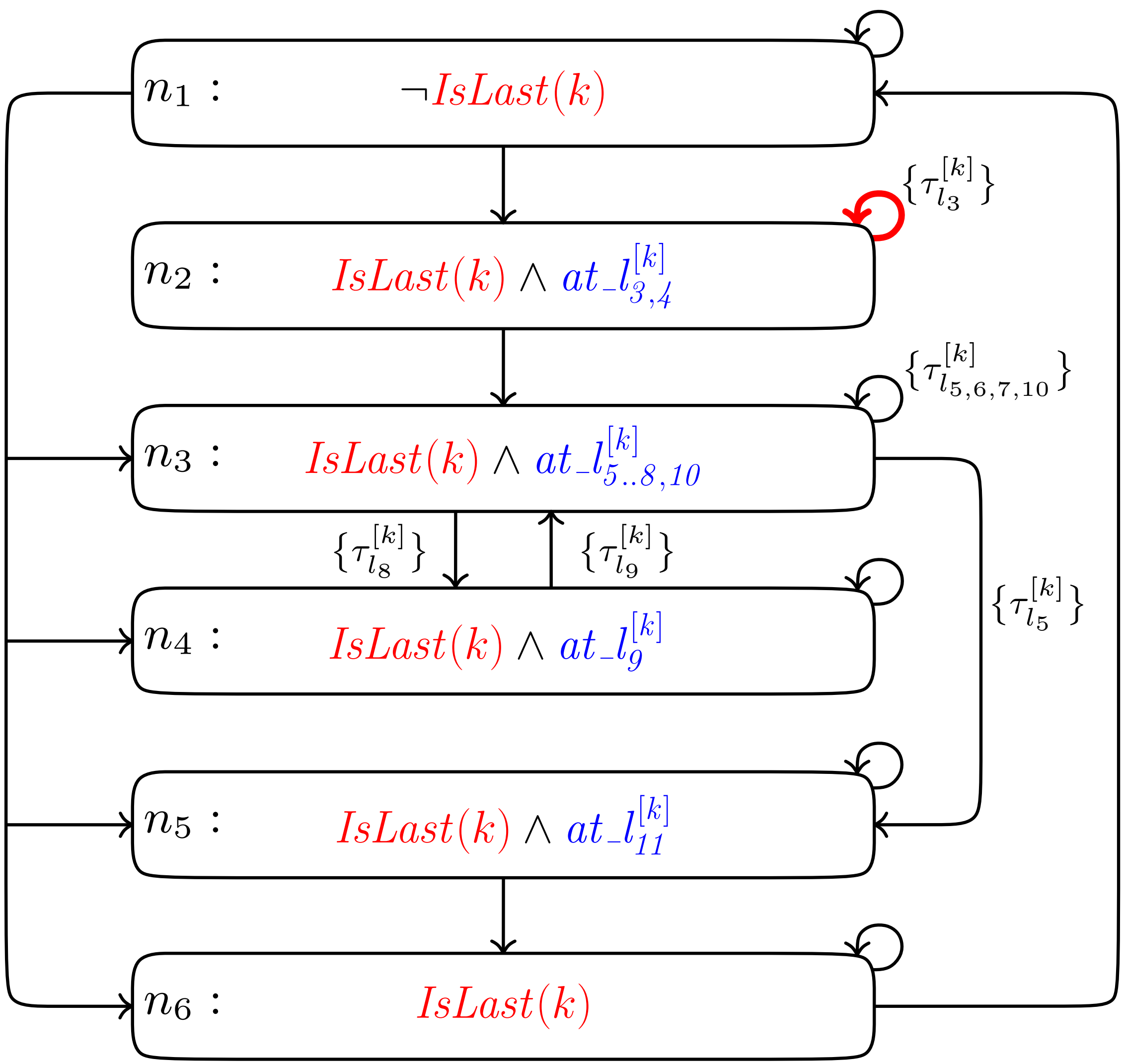


Reason about:

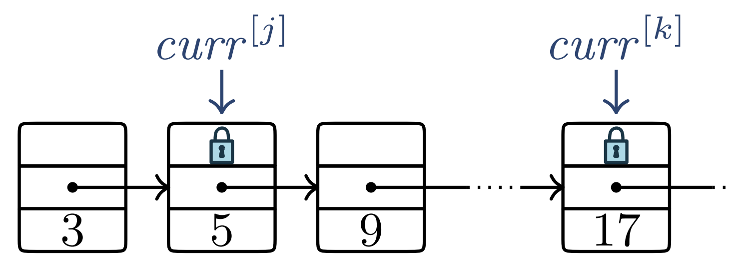
program position

$$\begin{pmatrix} IsLast(k) \wedge \ j \neq k \wedge \\ at\_l^{[k]}_{3,4} \wedge at\_l^{[j]}_{9} \wedge \\ curr^{[j]}.lockid = \emptyset \end{pmatrix} \wedge \ curr^{[j]}.lock(j) \rightarrow \begin{pmatrix} IsLast(k') \wedge at'\_l^{[k']}_{3,4} \wedge \\ at'\_l^{[j']}_{10} \wedge pres(V - curr^{[j]}) \wedge \\ curr'^{[j']}.lockid = j' \end{pmatrix}$$

Verification Diagram for "Last Terminates"



$$\mathcal{S}[N] \vDash \psi^{(k)}$$



Reason about:

reachability, cells, memory

$$\begin{pmatrix} IsLast(k) \wedge \ j \neq k \wedge \\ at\_l_{3,4}^{[k]} \wedge at\_l_{9}^{[j]} \wedge \\ curr^{[j]}.lockid = \emptyset \end{pmatrix} \wedge \ curr^{[j]}.lock(j) \rightarrow \begin{pmatrix} IsLast(k') \wedge at'\_l_{3,4}^{[k']} \wedge \\ at'\_l_{10}^{[j']} \wedge pres(V - curr^{[j]}) \wedge \\ curr'^{[j']}.lockid = j' \end{pmatrix}$$

Our Contribution

- **TLL3**, a theory for concurrent linked lists
- We show TLL3 **decidable**
- We propose a combination based **decision procedure**

TLL3: A Theory for Single-Linked Concurrent Lists

TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

Σ_{addr}

TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}}$

TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}}$

TLL3: A Theory for Single-Linked Concurrent Lists

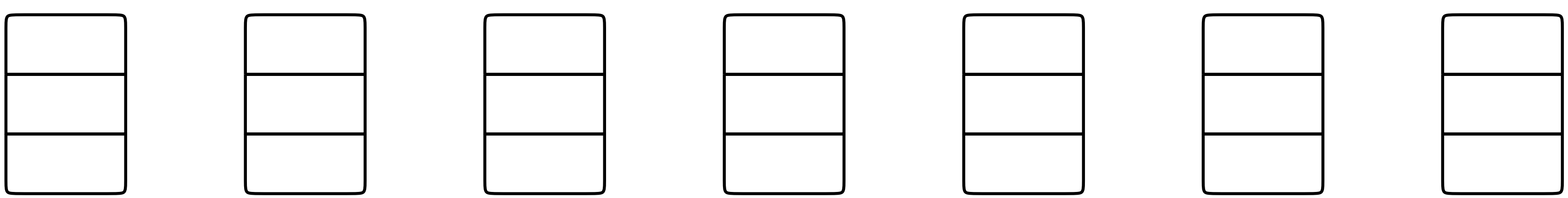
- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}}$$

TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

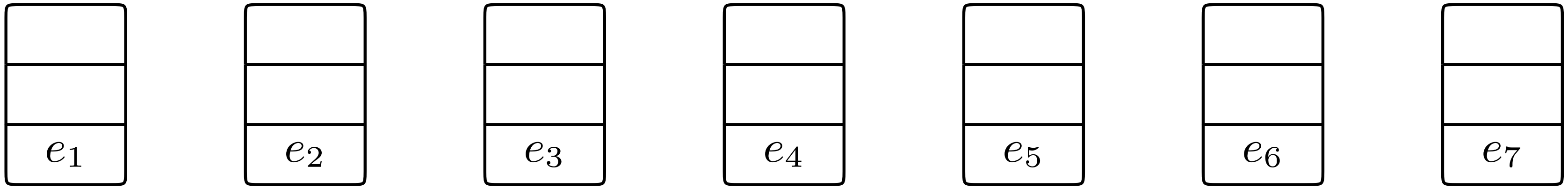
$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}}$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

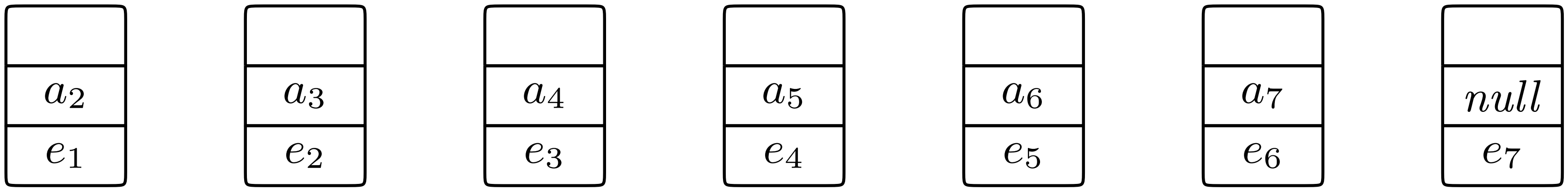
$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}}$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

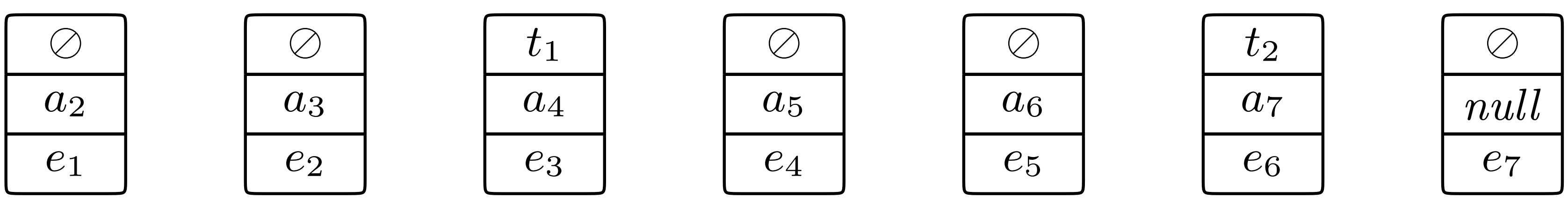
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

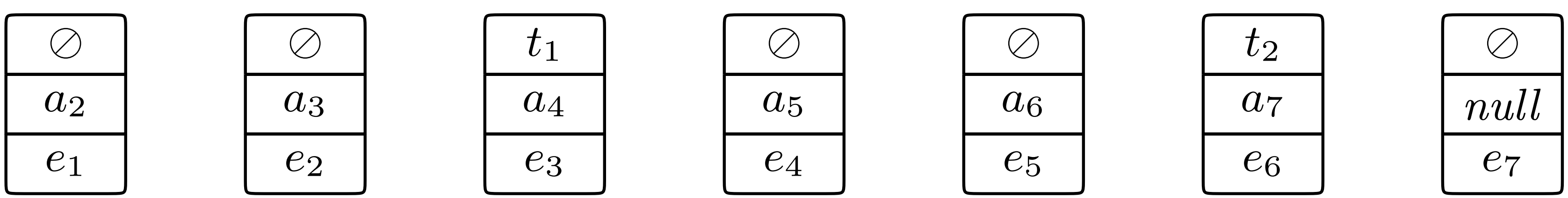
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

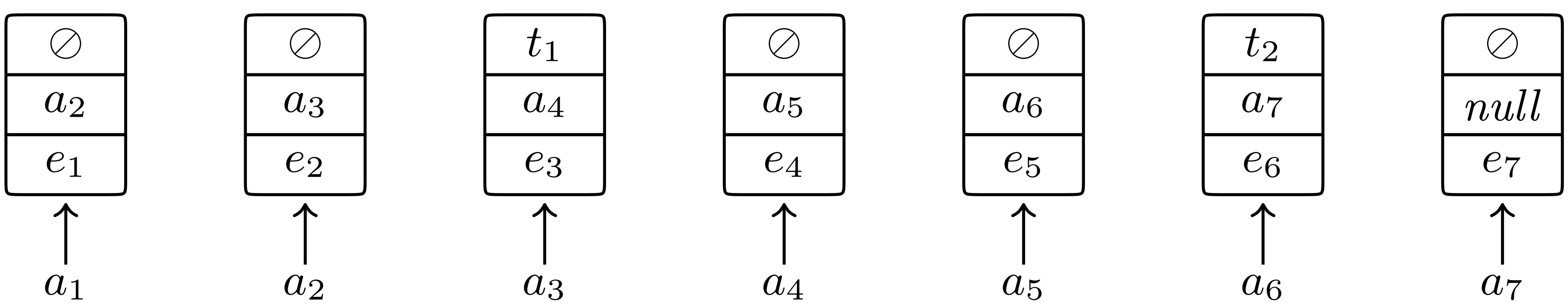
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

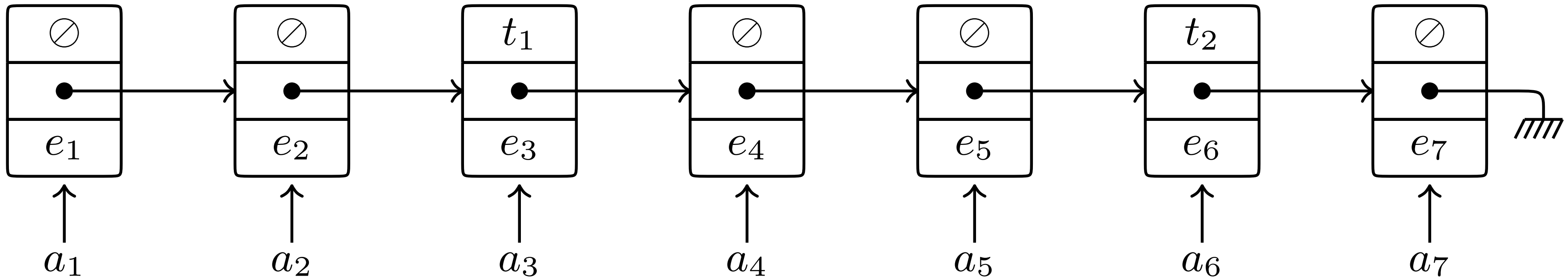
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

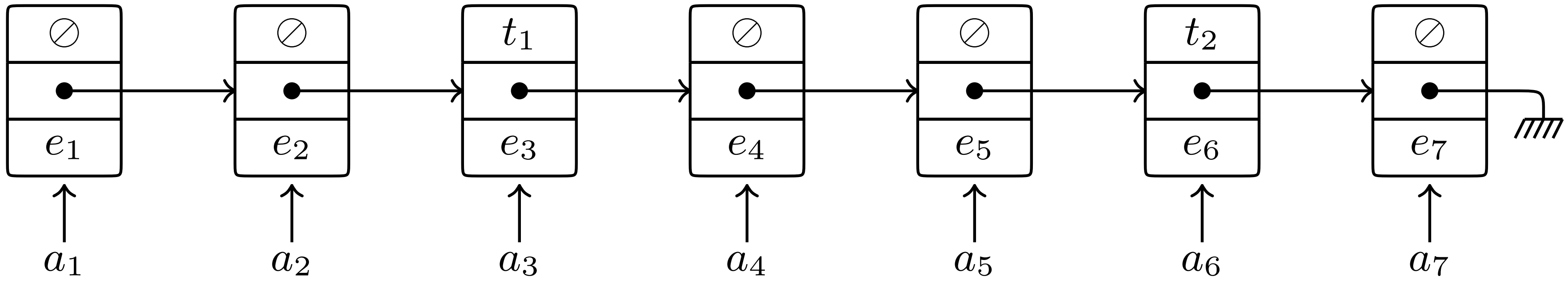
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

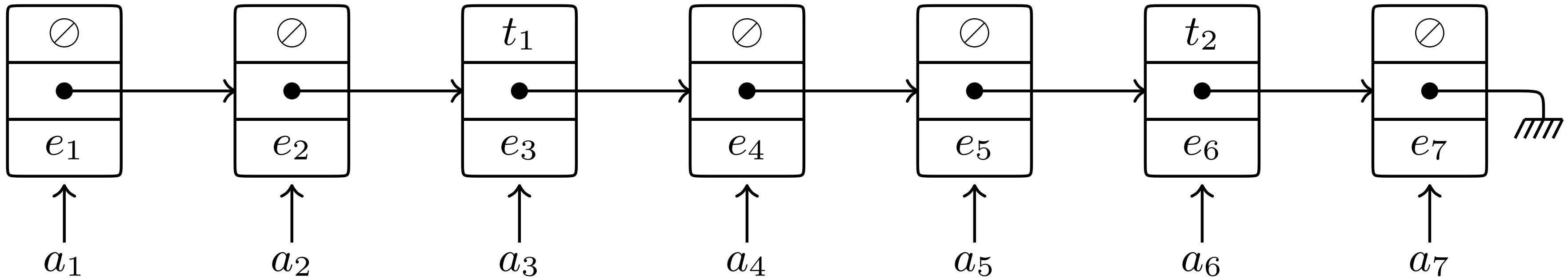
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

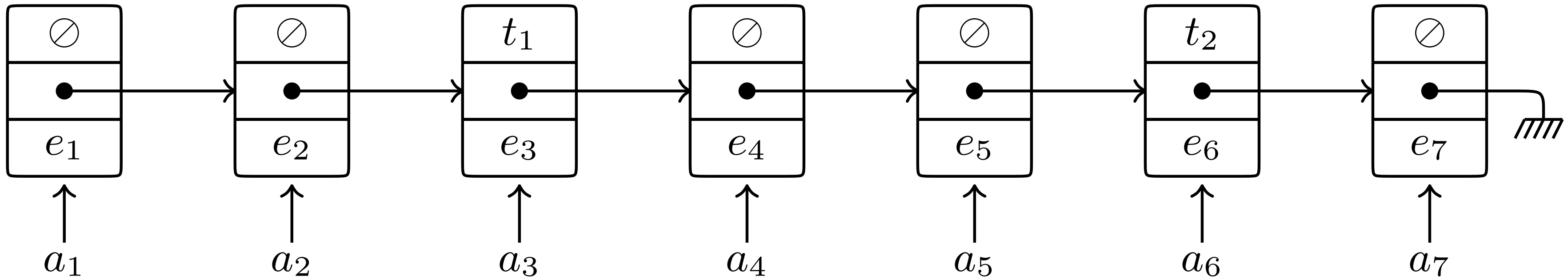
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

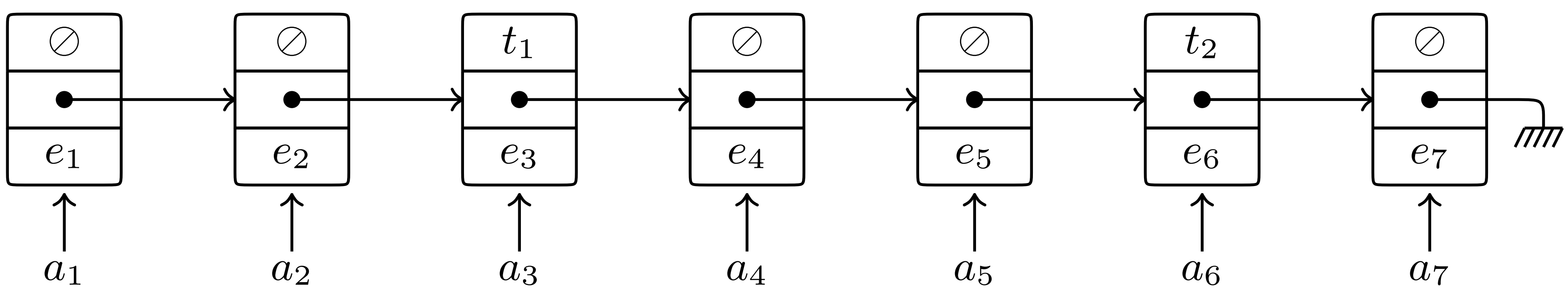


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

path $=$ a non-repeating sequence of addresses



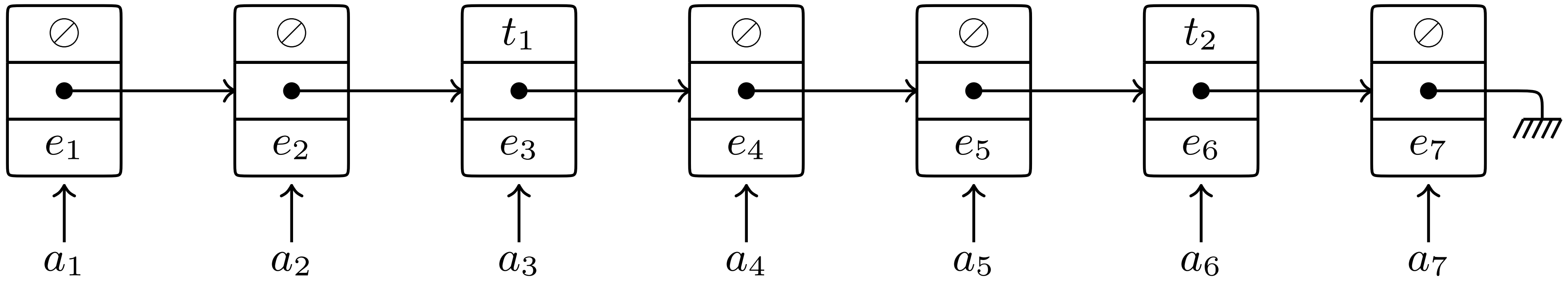
TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

path = a non-repeating sequence of addresses

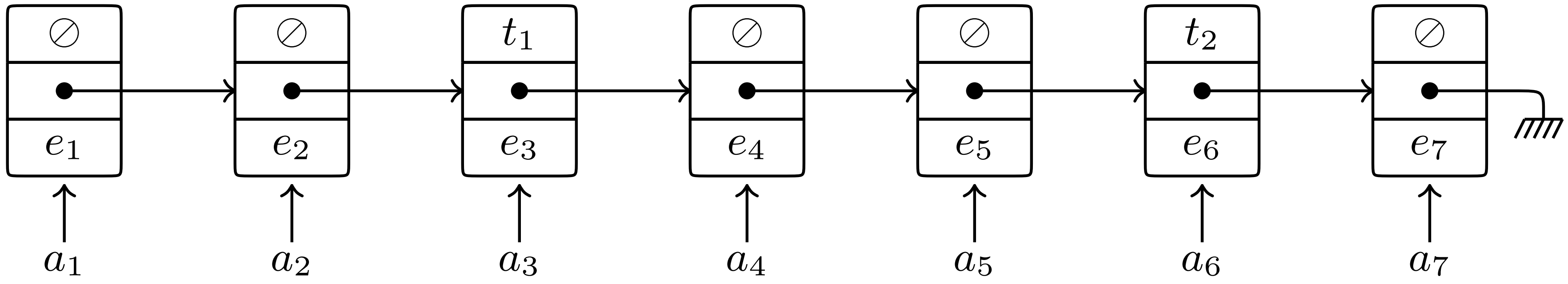
$$[a_1, a_2, a_3]$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

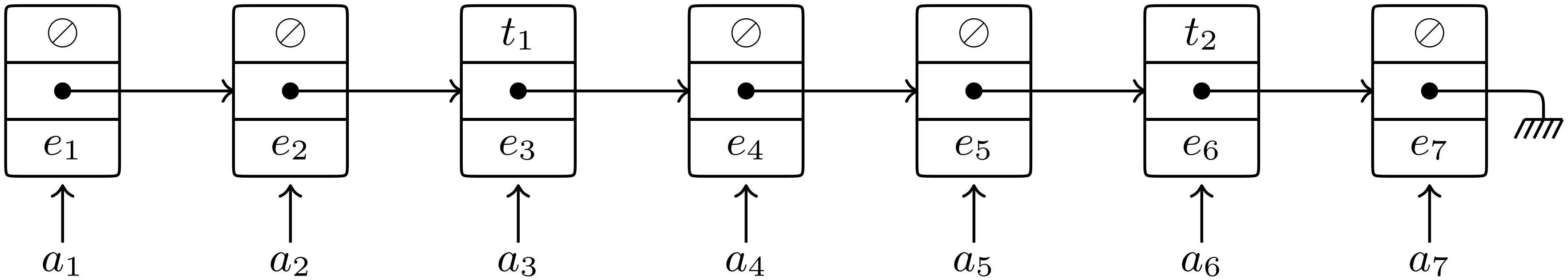


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

$$append([a_1, a_2], [a_3, a_4, a_5], [a_1, a_2, a_3, a_4, a_5])$$

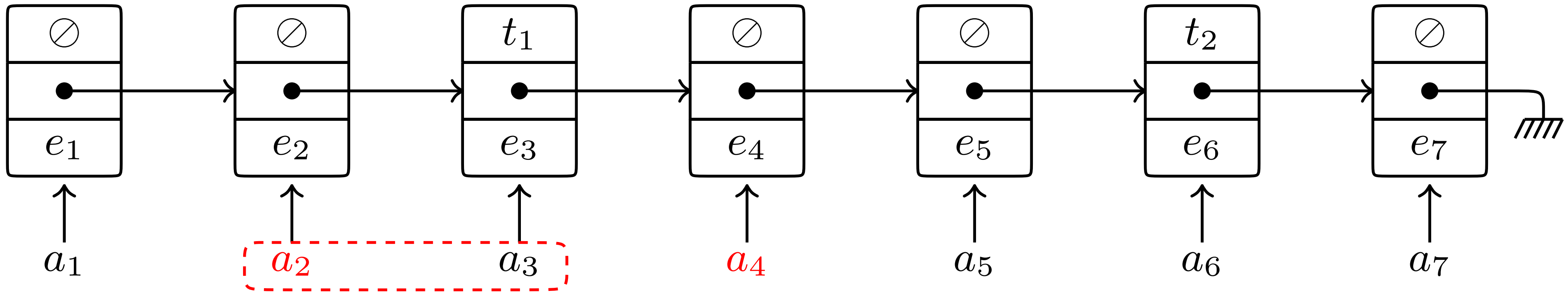


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$

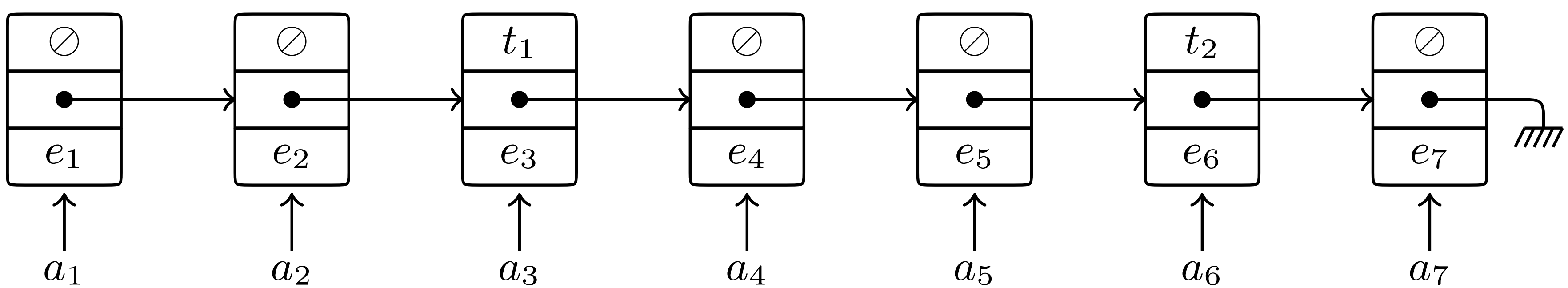
$$reach(m, a_2, a_4, [a_2, a_3])$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

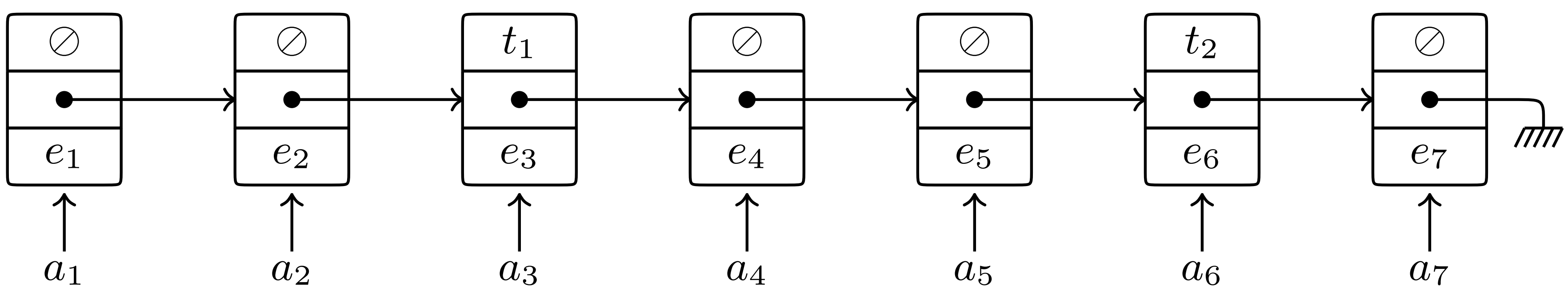
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}}$$



TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

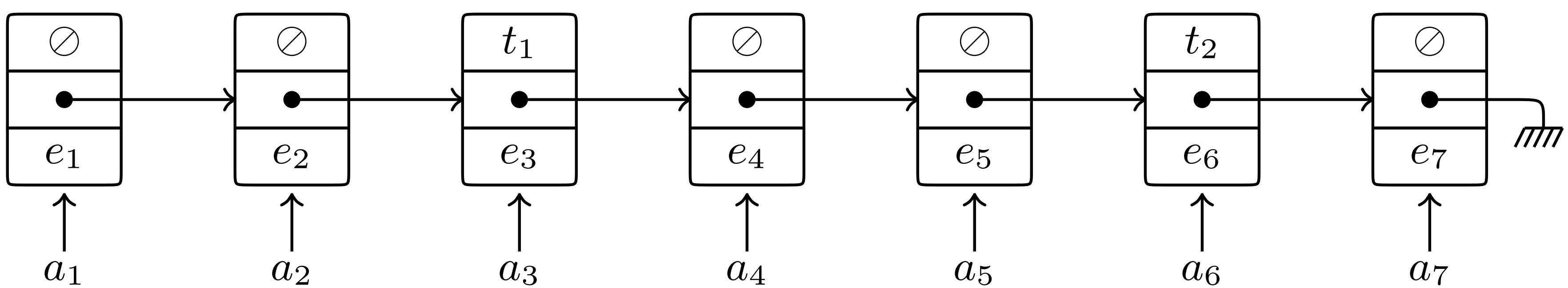


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$path2set([a_2, a_3]) = \{a_2, a_3\}$$

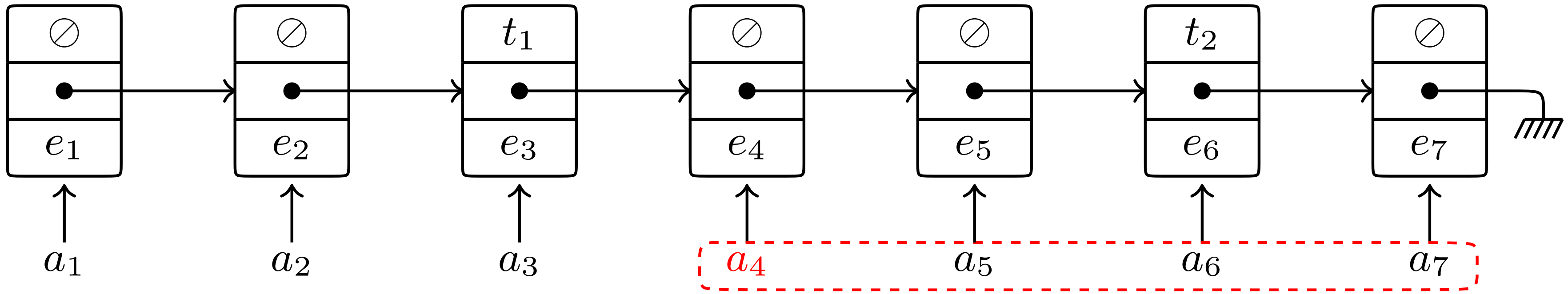


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{thid}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{setth}} \cup \Sigma_{\text{reachability}} \cup \Sigma_{\text{bridge}}$$

$$addr2set(m, a_4) = \{a_4, a_5, a_6, a_7\}$$

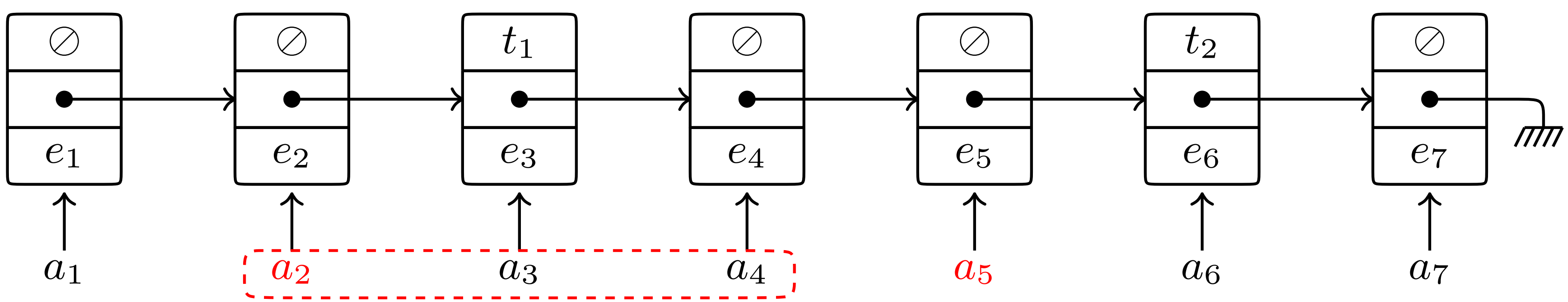


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$getp(m, a_2, a_5) = [a_2, a_3, a_4]$$

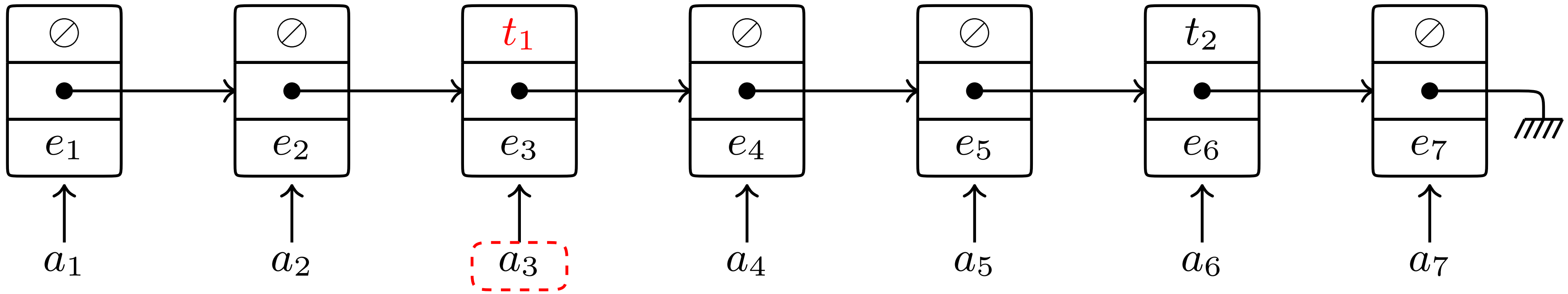


TLL3: A Theory for Single-Linked Concurrent Lists

- TLL3 is a union of other theories

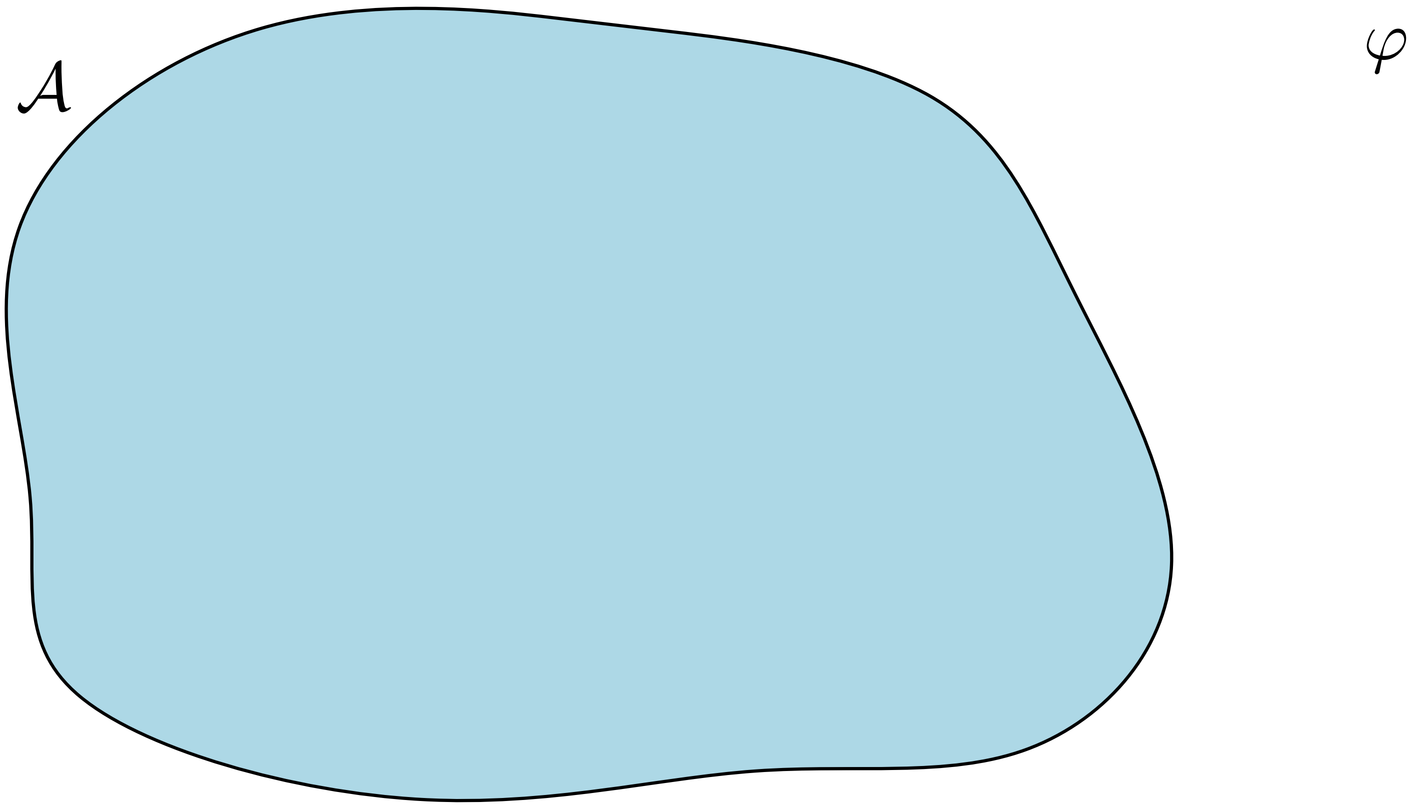
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{thid}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{setth}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$\mathit{firstlocked}(m, [a_1, a_2, a_3, a_4, a_5, a_6, a_7]) = a_3$$

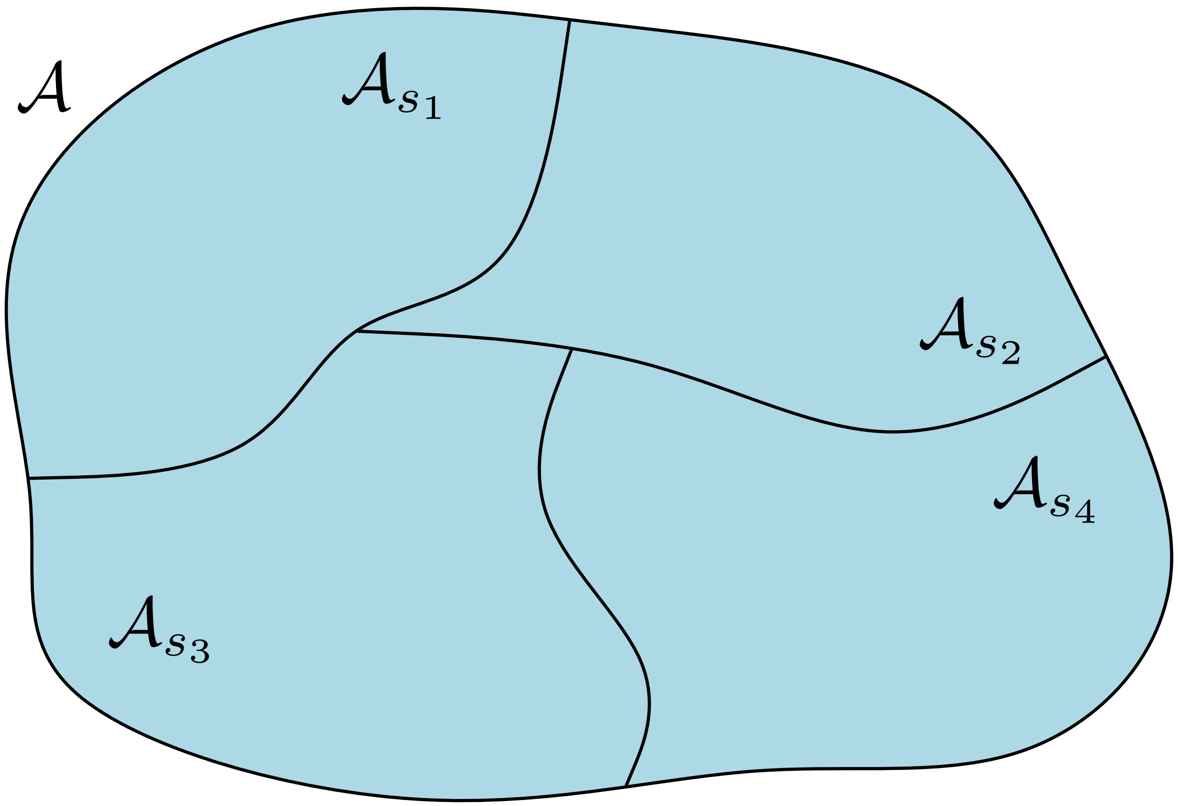


Decidability by Small Model Property (SMP)

Decidability by Small Model Property (SMP)

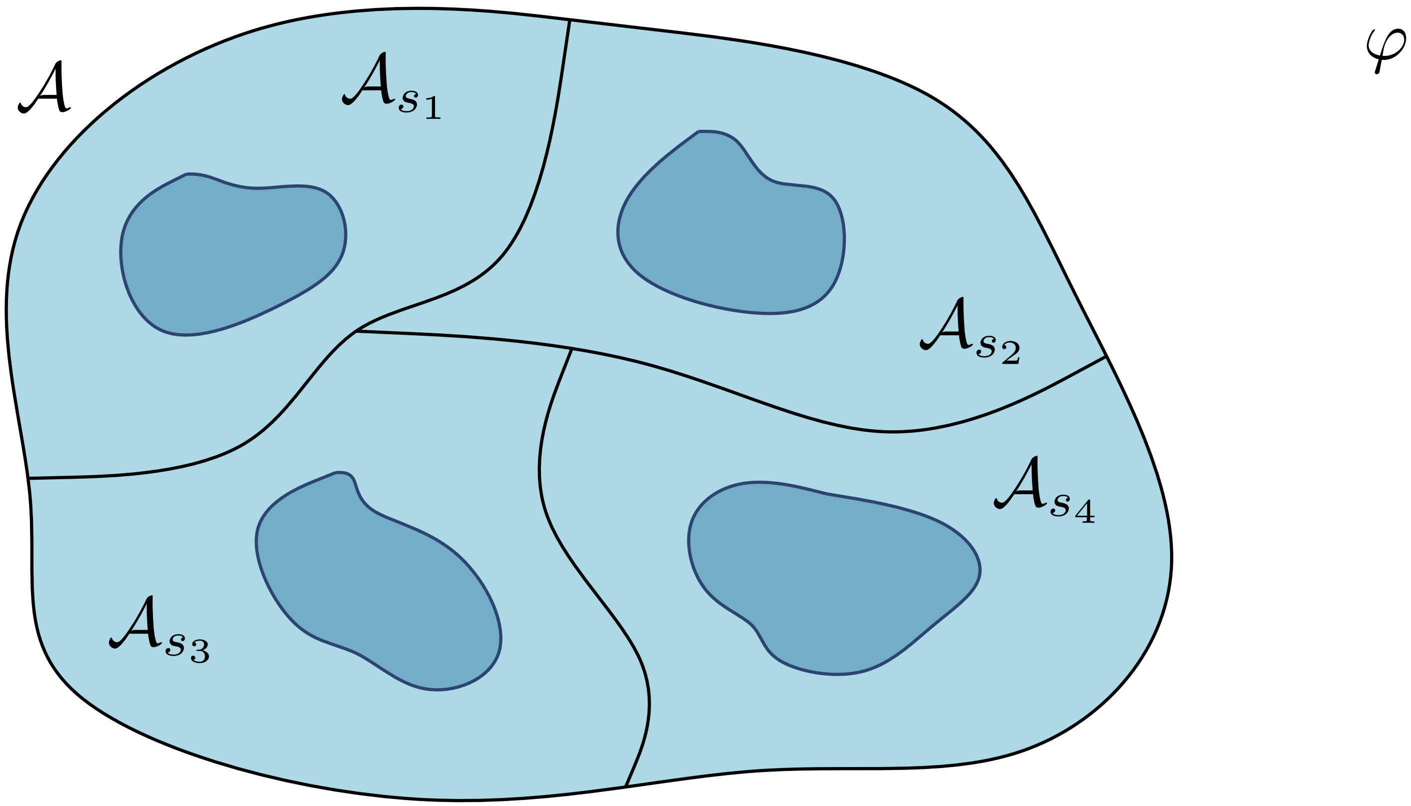


Decidability by Small Model Property (SMP)

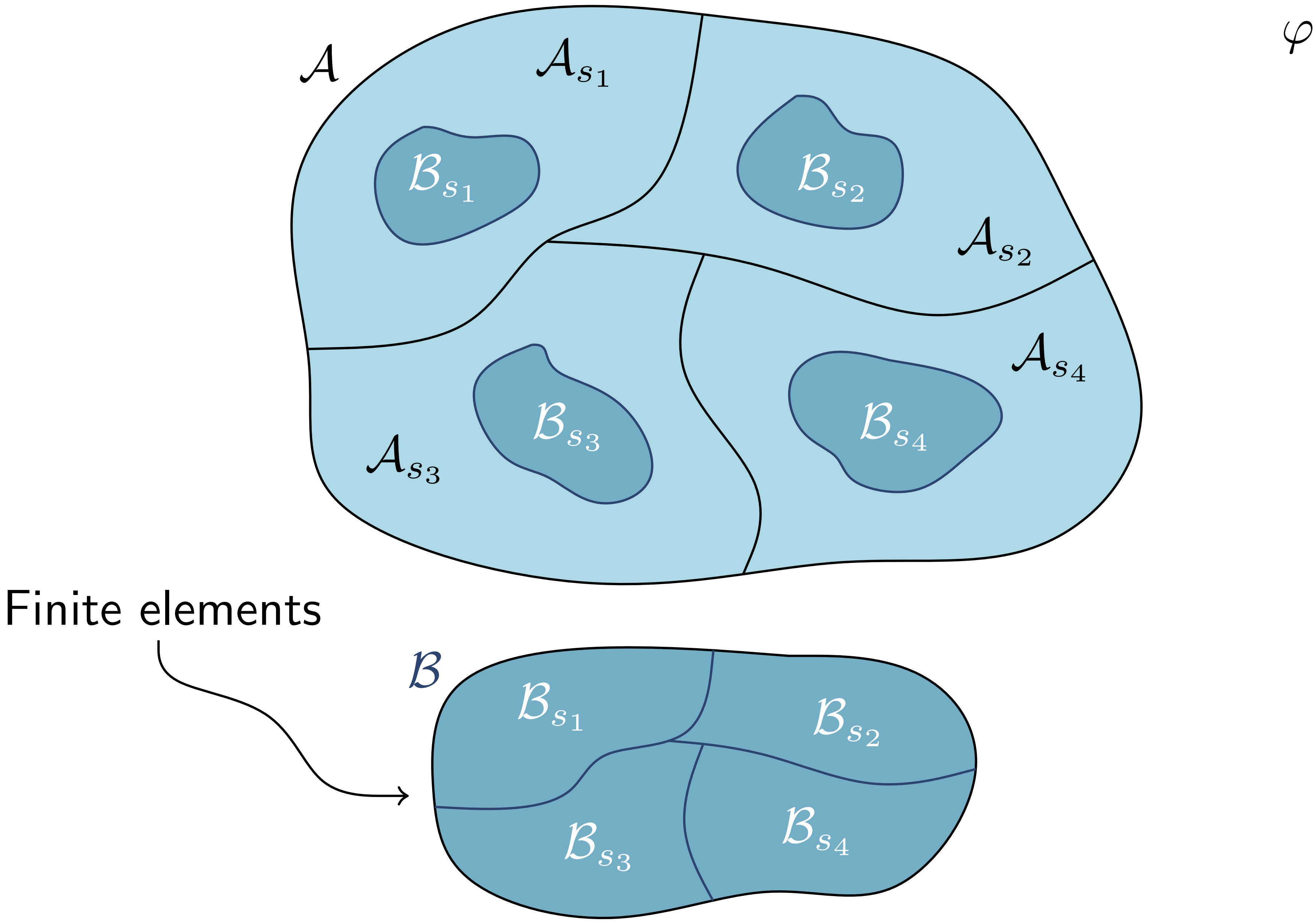


φ

Decidability by Small Model Property (SMP)

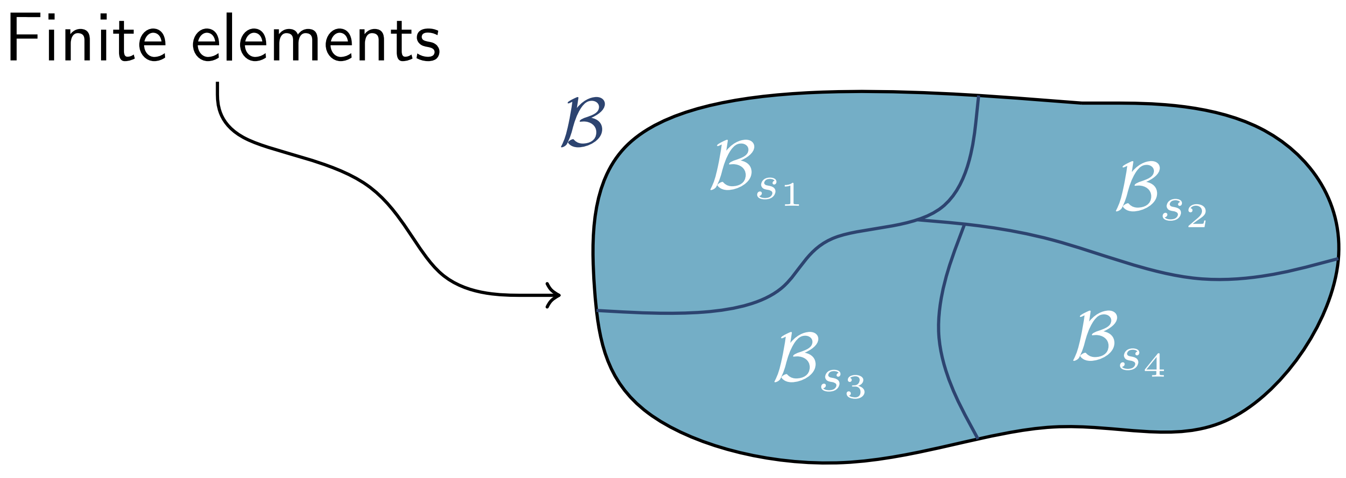


Decidability by Small Model Property (SMP)



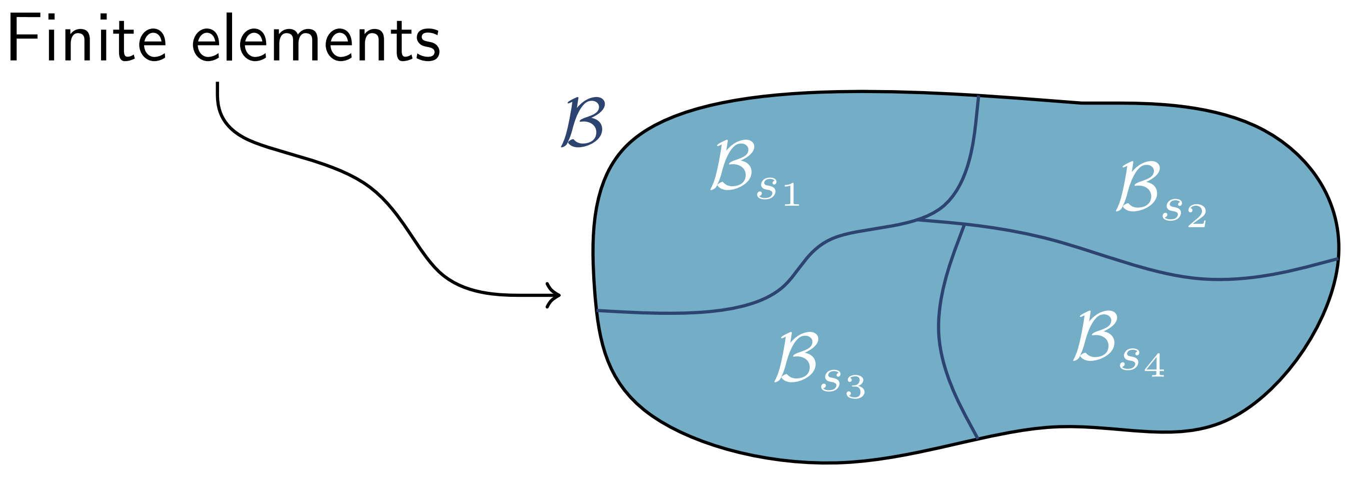
Decidability by Small Model Property (SMP)

- Let Γ be a conjunction of TLL3-literals



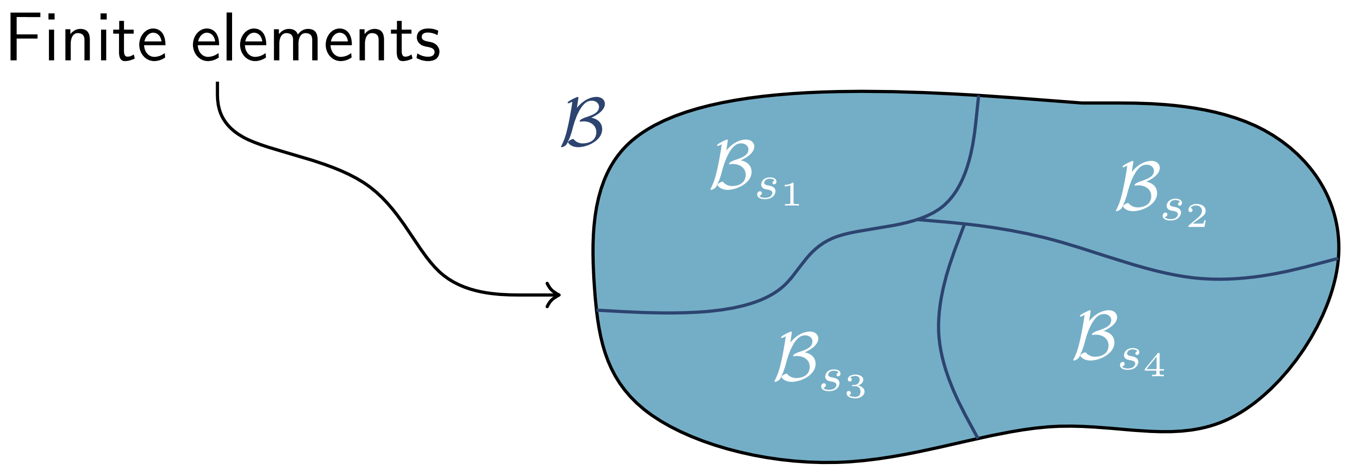
Decidability by Small Model Property (SMP)

- Let Γ be a conjunction of TLL3-literals
- If Γ is satisfied in an arbitrary TLL3 interpretation $\mathcal{A}$



Decidability by Small Model Property (SMP)

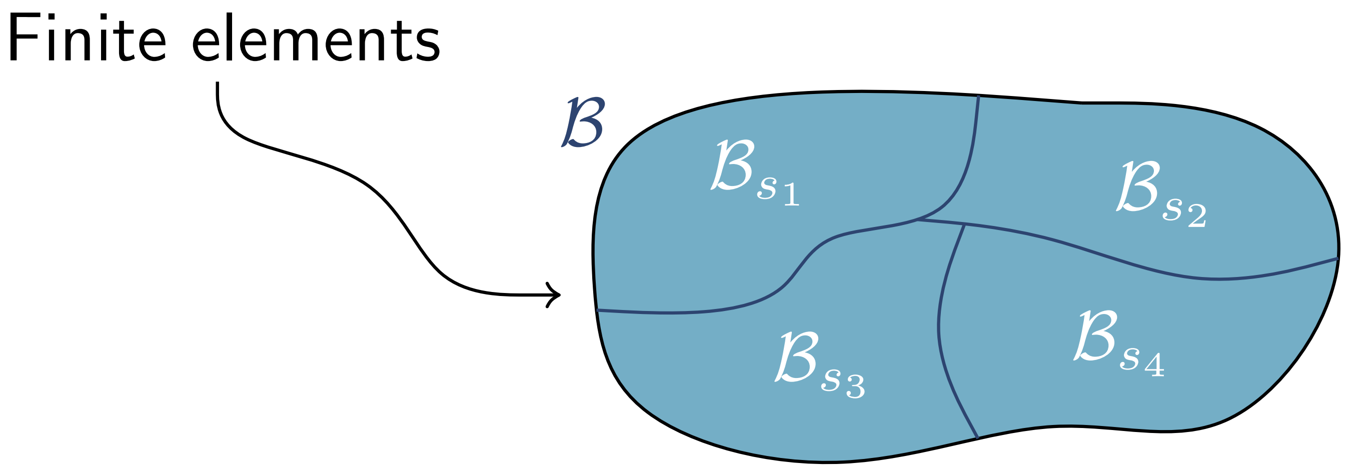
- Let Γ be a conjunction of TLL3-literals
- If Γ is satisfied in an arbitrary TLL3 interpretation $\mathcal{A}$
- We construct a new finite interpretation $\mathcal{B}$ **bounded by** Γ



Decidability by Small Model Property (SMP)

- Let Γ be a conjunction of TLL3-literals
- If Γ is satisfied in an arbitrary TLL3 interpretation $\mathcal{A}$
- We construct a new finite interpretation $\mathcal{B}$ **bounded by** Γ

TLL3 is **decidable** by enumerating all possible elements



A Decision Procedure for TLL3 (main idea)

A Decision Procedure for TLL3 (main idea)

- We use a **many-sorted variant of Nelson-Oppen**

A Decision Procedure for TLL3 (main idea)

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

A Decision Procedure for TLL3 (main idea)

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory

A Decision Procedure for TLL3 (main idea)

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory

- Theories **share only sorts**

A Decision Procedure for TLL3 (main idea)

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory
- Theories **share only sorts**
- When two theories are combined:
  - Both are **stable infinite**
  - One is **polite** to the other with respect to shared sorts

Stable Infinite & Politeness

Stable Infinite & Politeness

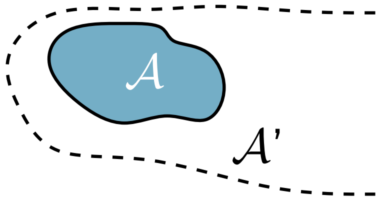
- **Stable infinite**

- **Polite** with respect to sorts $s_1, \ldots s_n$

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}$'

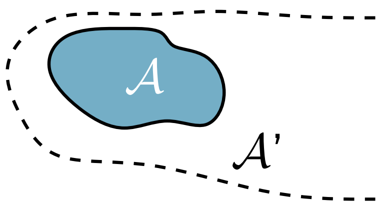


- **Polite** with respect to sorts $s_1, \ldots s_n$

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}$'

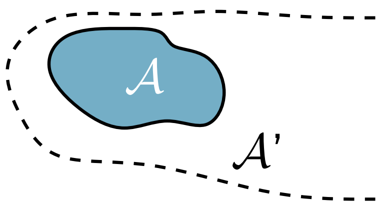


- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**
  - **Finite witnessable**

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}$'



- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**

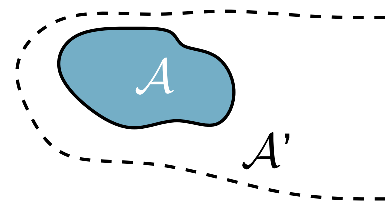
    if $\mathcal{A}$ is a model of φ, with domains $\mathcal{A}_{s_1} \quad \cdots \quad \mathcal{A}_{s_n}$

  - **Finite witnessable**

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}'$



- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**

    if $\mathcal{A}$ is a model of φ, with domains $|\mathcal{A}_{s_1}| \cdots |\mathcal{A}_{s_n}|$

    then, for every $k_1 \cdots k_n$

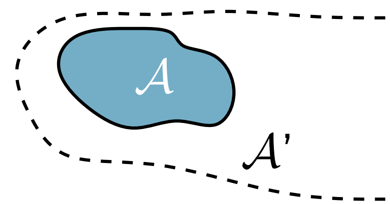
    $$\begin{array}{ccc} |\mathcal{A}_{s_1}| & \cdots & |\mathcal{A}_{s_n}| \\ \leq & & \leq \\ k_1 & \cdots & k_n \end{array}$$

  - **Finite witnessable**

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}$'



- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**

    if $\mathcal{A}$ is a model of φ, with domains

    then, for every

    there is a model $\mathcal{B}$ of φ, such that

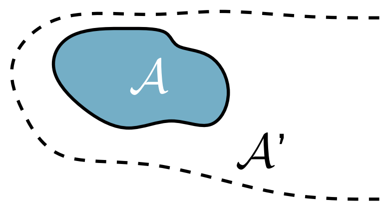
$$\begin{array}{ccc} |\mathcal{A}_{s_1}| & \cdots & |\mathcal{A}_{s_n}| \\ \geq & & \geq \\ k_1 & \cdots & k_n \\ = & & = \\ |\mathcal{B}_{s_1}| & \cdots & |\mathcal{B}_{s_n}| \end{array}$$

  - **Finite witnessable**

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}'$



- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**

    if $\mathcal{A}$ is a model of φ, with domains $|\mathcal{A}_{s_1}| \cdots |\mathcal{A}_{s_n}|$

    then, for every $k_1 \cdots k_n$

    there is a model $\mathcal{B}$ of φ, such that $|\mathcal{B}_{s_1}| \cdots |\mathcal{B}_{s_n}|$

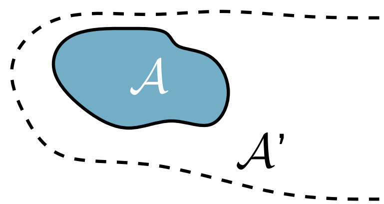
    $$\begin{array}{ccc} |\mathcal{A}_{s_1}| & \cdots & |\mathcal{A}_{s_n}| \\ \geq & & \geq \\ k_1 & \cdots & k_n \\ = & & = \\ |\mathcal{B}_{s_1}| & \cdots & |\mathcal{B}_{s_n}| \end{array}$$
  - **Finite witnessable**

    f

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}$'



- **Polite** with respect to sorts $s_1, \ldots s_n$
  - **Smooth**

    if $\mathcal{A}$ is a model of φ, with domains $|\mathcal{A}_{s_1}| \cdots |\mathcal{A}_{s_n}|$

    $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\;\; \not\geq \qquad\quad \not\geq$

    then, for every $k_1 \cdots k_n$

    $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\;\; || \qquad\quad ||$

    there is a model $\mathcal{B}$ of φ, such that $|\mathcal{B}_{s_1}| \cdots |\mathcal{B}_{s_n}|$
  - **Finite witnessable**

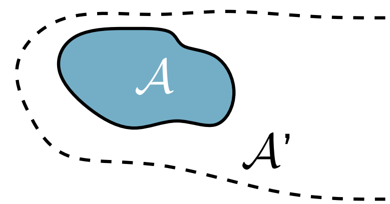
    f

    φ

Stable Infinite & Politeness

- **Stable infinite**

  for all QF-formula φ, exists an infinite interpretation $\mathcal{A}'$



- **Polite** with respect to sorts $s_1, \dots s_n$
  - **Smooth**

    if $\mathcal{A}$ is a model of φ, with domains $|\mathcal{A}_{s_1}| \cdots |\mathcal{A}_{s_n}|$

    $$\begin{array}{ccc} |\mathcal{A}_{s_1}| & \cdots & |\mathcal{A}_{s_n}| \\ \not\wedge & & \not\wedge \\ k_1 & \cdots & k_n \\ || & & || \\ |\mathcal{B}_{s_1}| & \cdots & |\mathcal{B}_{s_n}| \end{array}$$

    then, for every $k_1 \cdots k_n$

    there is a model $\mathcal{B}$ of φ, such that $|\mathcal{B}_{s_1}| \cdots |\mathcal{B}_{s_n}|$
  - **Finite witnessable**

    $$\varphi \xrightarrow{\quad f \quad} (\exists \bar{v})\psi \quad \text{s.t. } [\bar{v} = V_\varphi \setminus V_\psi]$$

if ψ is **satisfiable**, then exists $\mathcal{B}$ with **one variable per value**

Checklist

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory
- Theories **share only sorts**
- When two theories are combined:
  - Both are **stable infinite**
  - One is **polite** to the other with respect to shared sorts

Checklist

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory ✓ (except $T_{\text{reachability}}$ and Σ_{bridge})
- Theories **share only sorts**
- When two theories are combined:
  - Both are **stable infinite**
  - One is **polite** to the other with respect to shared sorts

Checklist

- We use a **many-sorted variant of Nelson-Oppen**

To apply it, we require:

- A **decision procedure** for each theory ✓ (except $T_{\text{reachability}}$ and Σ_{bridge})
- Theories **share only sorts** ✓
- When two theories are combined:
  - Both are **stable infinite**
  - One is **polite** to the other with respect to shared sorts

A Combination-based Decision Procedure for TLL3

A Combination-based Decision Procedure for TLL3

$$\mathsf{TLL3} = \cdots \oplus T_{\mathsf{reachability}} \oplus \cdots \oplus \textit{bridge functions and predicates}$$

A Combination-based Decision Procedure for TLL3

$$\mathsf{TLL3} = \cdots \oplus T_{\mathsf{reachability}} \oplus \cdots \oplus \textit{bridge functions and predicates}$$

$$T_{\mathsf{Base}} \;=\; T_{\mathsf{addr}} \oplus T_{\mathsf{elem}} \oplus T_{\mathsf{thid}} \oplus T_{\mathsf{cell}} \oplus T_{\mathsf{mem}} \oplus T_{\mathsf{fseq}} \oplus T_{\mathsf{set}} \oplus T_{\mathsf{setth}}$$

A Combination-based Decision Procedure for TLL3

$$\mathsf{TLL3} = \cdots \oplus T_{\mathsf{reachability}} \oplus \cdots \oplus \textit{bridge functions and predicates}$$

$$T_{\mathsf{Base}} = T_{\mathsf{addr}} \oplus T_{\mathsf{elem}} \oplus T_{\mathsf{thid}} \oplus T_{\mathsf{cell}} \oplus T_{\mathsf{mem}} \oplus \overbrace{T_{\mathsf{fseq}}}^{T_{\mathsf{path}} \xleftarrow{PATH}} \oplus T_{\mathsf{set}} \oplus T_{\mathsf{setth}}$$

A Combination-based Decision Procedure for TLL3

$\mathsf{TLL3} = \cdots \oplus T_{\mathsf{reachability}} \oplus \cdots \oplus$ *bridge functions and predicates*

$\widehat{\mathsf{TLL3}}$

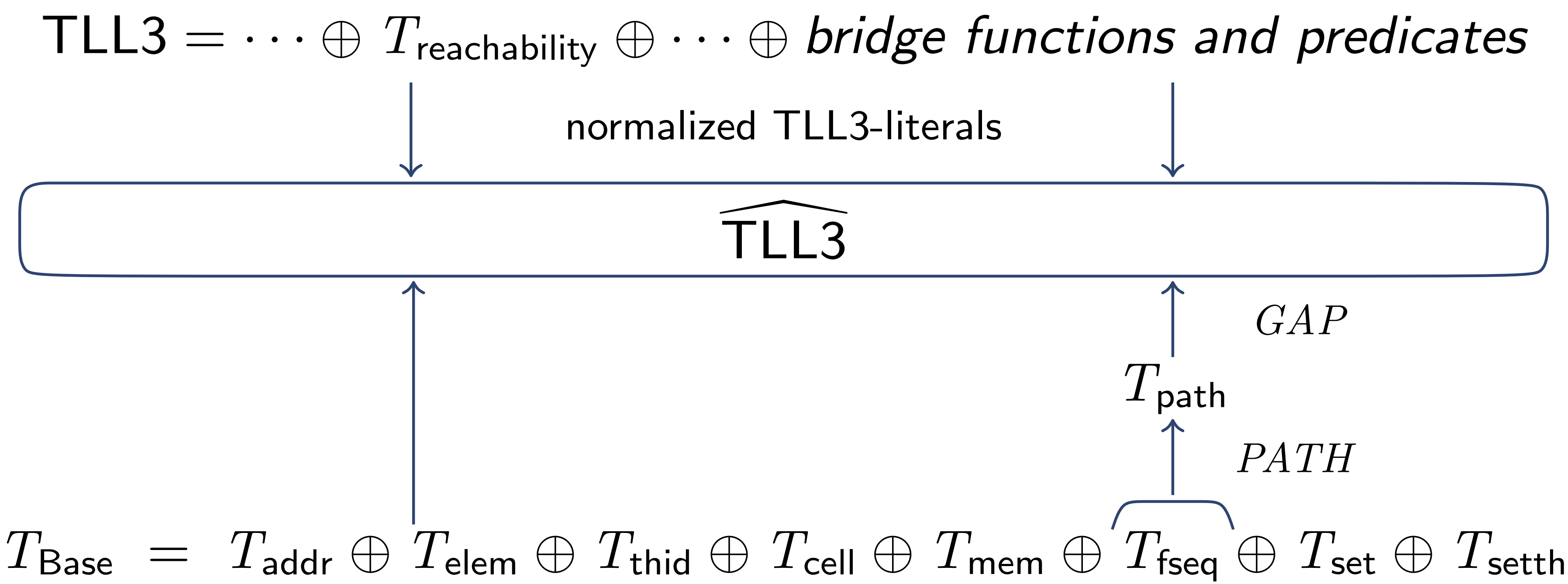
GAP

T_{path}

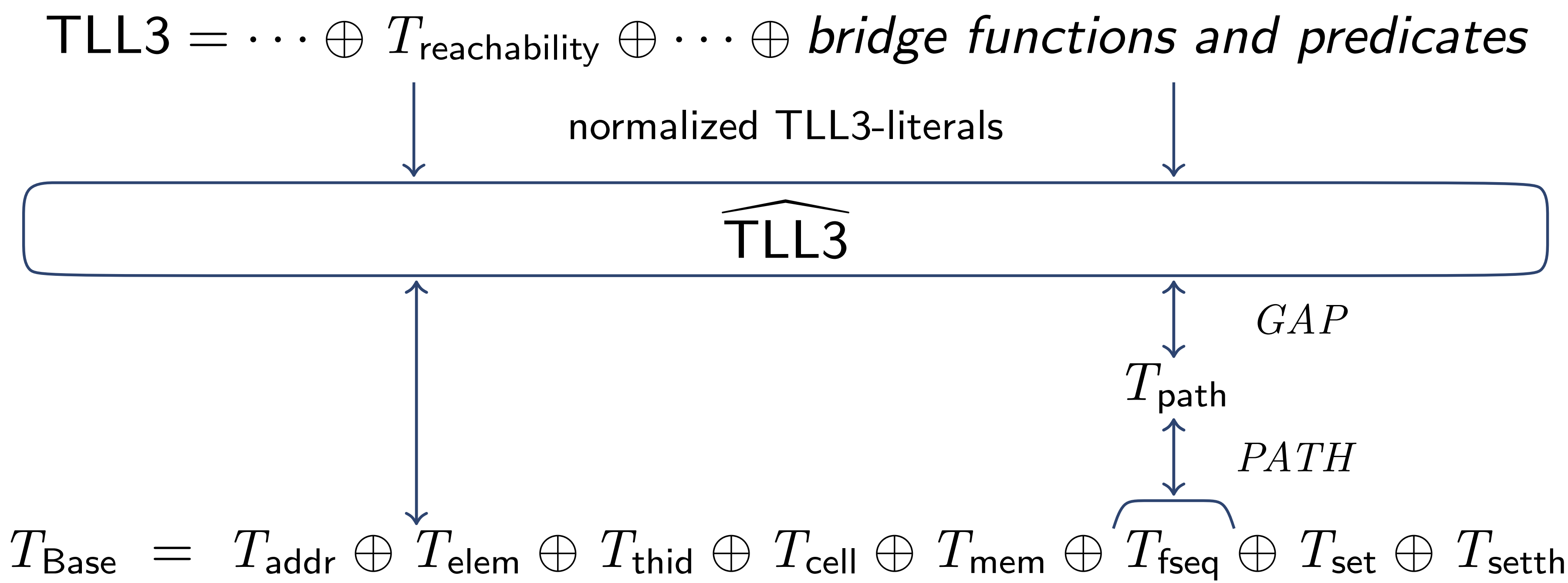
$PATH$

$$T_{\mathsf{Base}} \;=\; T_{\mathsf{addr}} \oplus T_{\mathsf{elem}} \oplus T_{\mathsf{thid}} \oplus T_{\mathsf{cell}} \oplus T_{\mathsf{mem}} \oplus T_{\mathsf{fseq}} \oplus T_{\mathsf{set}} \oplus T_{\mathsf{setth}}$$

A Combination-based Decision Procedure for TLL3

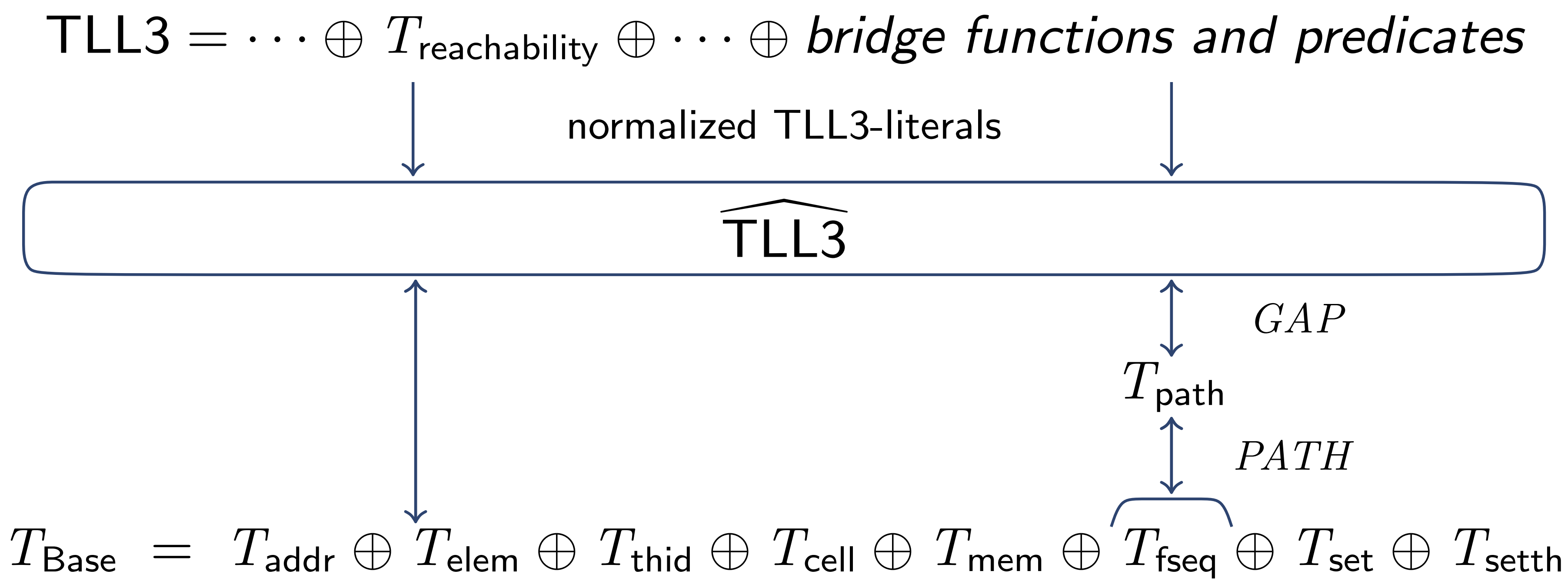


A Combination-based Decision Procedure for TLL3



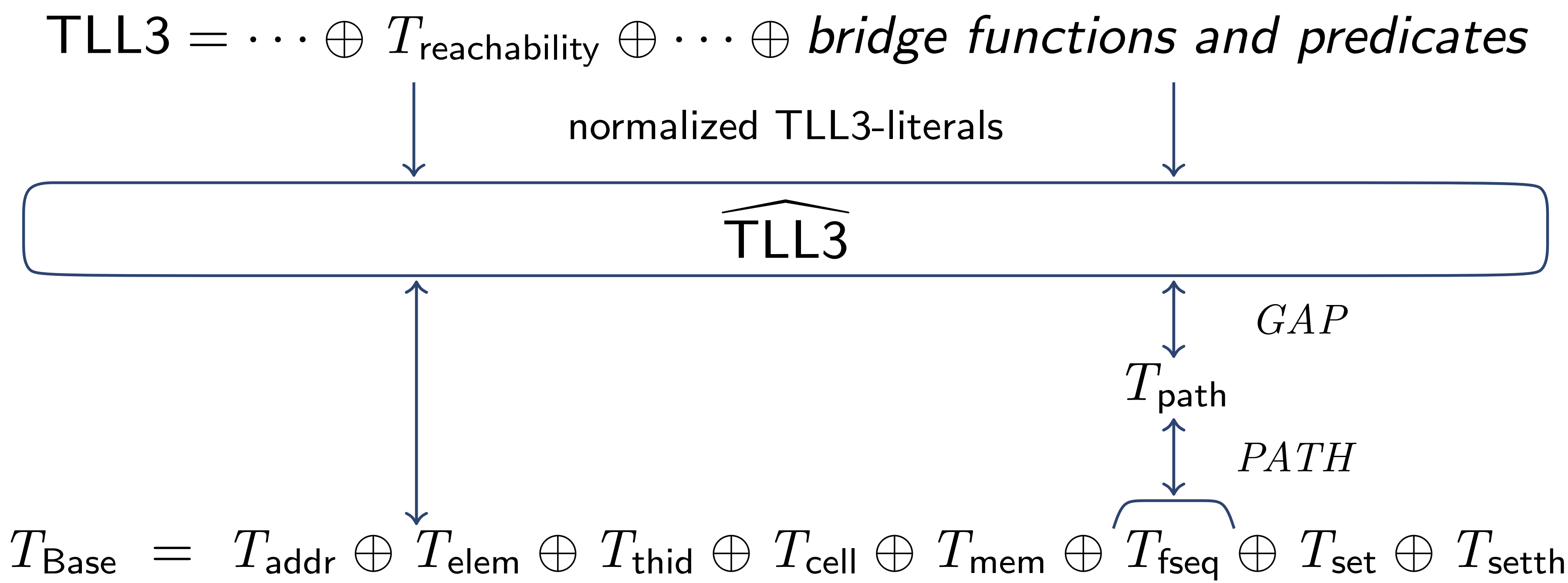
- **Unfolding** of definitions in $PATH$ and GAP

A Combination-based Decision Procedure for TLL3



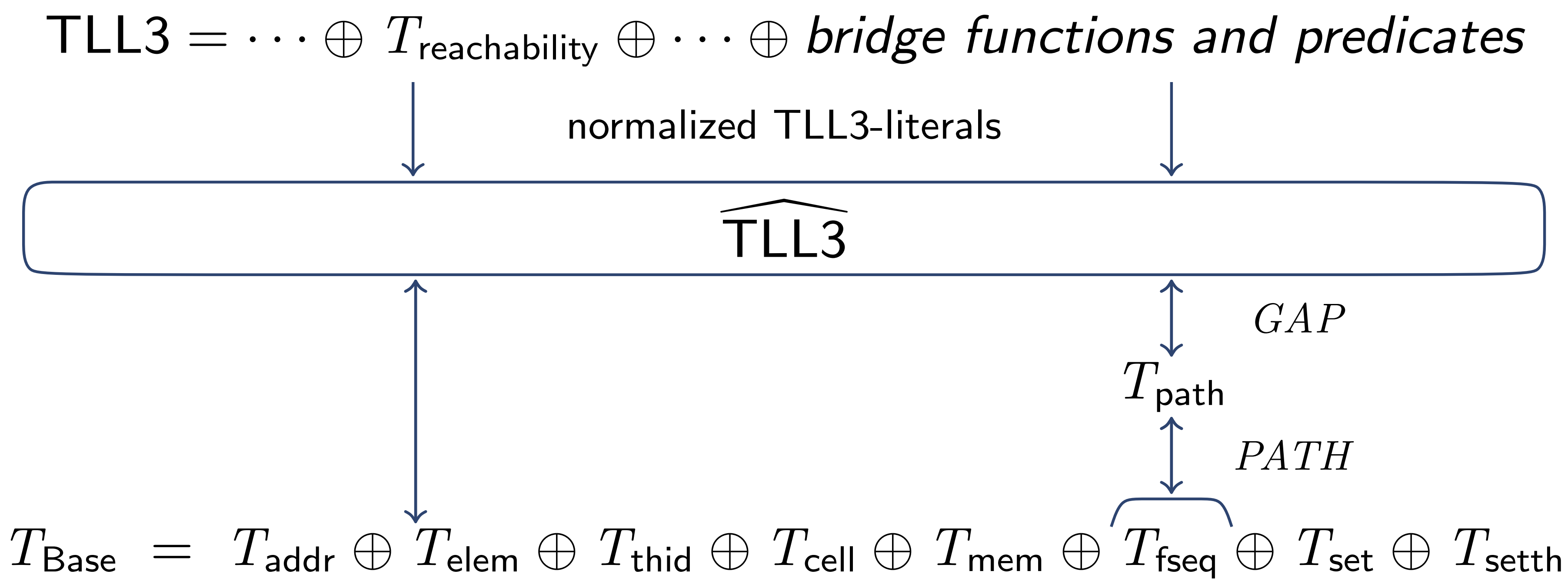
- **Unfolding** of definitions in $PATH$ and GAP
- *SMP* guarantees that all theories are **finite witnessable**

A Combination-based Decision Procedure for TLL3



- **Unfolding** of definitions in $PATH$ and GAP
- *SMP* guarantees that all theories are **finite witnessable**
- T_{cell}, T_{mem}, T_{set}, T_{setth} and T_{fseq} are all **stable infinite**

A Combination-based Decision Procedure for TLL3



- **Unfolding** of definitions in $PATH$ and GAP
- *SMP* guarantees that all theories are **finite witnessable**
- T_{cell}, T_{mem}, T_{set}, T_{setth} and T_{fseq} are all **stable infinite**
- These are **smooth** with respect to sorts addr, elem and thid

Conclusions

- We defined **TLL3**, a theory for concurrent single-linked lists
- We proved TLL3 **decidable**, by *Small Model Property*
- We provide a **combination-based decision procedure** for TLL3
- A step towards the assisted verification of **temporal properties** over **concurrent data-types**: VD + DP
- **Current and future** work:
  parametrized verification diagrams, DP for concurrent skiplists, concurrent hash-maps, concurrent Schorr-Waite
- Many possible **collaborations**:
  DPs as combination, SMTs, implementation