A Decision Procedure for Skiplists with Unbounded Height and Length

Alejandro Sánchez\textsuperscript{1} \hspace{1cm} César Sánchez\textsuperscript{1,2}

\textsuperscript{1}IMDEA Software Institute, Spain
\textsuperscript{2}Spanish Council for Scientific Research (CSIC), Spain

SVARM’11, Tallinn, 31 March 2012
Skiplists
Skiplists

- Sorted list of elements
Skiplists

- Sorted list of elements
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
SkipLists

- Sorted list of elements
- Hierarchy of linked lists

![Diagram of SkipLists]

-∞ → 2 → 5 → 7 → 10 → 12 → 17 → +∞
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
Skiplists

- Sorted list of elements
- Hierarchy of linked lists

class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}

head −∞ 2 5 7 10 12 17 +∞
tail
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}
```

```cpp
class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

![Diagram of a skiplist]
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert*(14) with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert*(14) with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```java
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

`insert(14)` with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

```
insert(14) with height 1
```

![Diagram of skiplist structure](image.png)
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

[Diagram showing a skiplist with elements 0 to 17, head and tail nodes, and insert(14) with height 1 highlighted.]
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

`insert(14)` with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

`insert(14)` with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}
```

```cpp
class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

**insert(14)** with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
};

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
};
```

Example:
```
insert(14) with height 1
```
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```java
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert(14) with height 1*
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

`insert(14)` with height 1
**Skiplists**

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert(14) with height 1*
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```java
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

`insert(14)` with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

**insert(14) with height 1**
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

**insert(14) with height 1**
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
};

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
};
```

`insert(14)` with height 1
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```java
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert(14) with height 1*
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

**insert(14) with height 1**
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert(14) with height 1*
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

(insert(14) with height 1)
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}

insert(14) with height 1
**Skiplists**

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```java
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

**insert(14) with height 1**
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```cpp
class Skiplist {
    Node* head;
    Node* tail;
    int maxLevel;
}

class Node {
    Value v;
    Key k;
    Array<Node*>(4) next;
}
```

*insert(14) with height 1*
Why skiplists of unbounded height and length?
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $TSL_K$
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $TSL_K$
  - Show it decidable by finite model property
  - Arbitrary length...
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $\text{TSL}_K$
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $\text{TSL}_K$
- Show it decidable by finite model property
- Arbitrary length...
- ... but bounded height!
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $\text{TSL}_K$
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $\text{TSL}_K$
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!

In practice, \textit{performance is lost!}
Why skiplists of unbounded height and length?

- While visited EPFL, we developed TSL
- Show it decidable by finite model property
- Arbitrary length...
- ... but bounded height!

In practice, performance is lost!

Dynamic height is required
Verification of Skiplists

\[ max_H = 3 \]
Verification of Skiplists

**Skiplist shape preservation**: $\square \text{SkipList}(h, sl, r)$

$\text{SkipList}(h, sl, r) \equiv$

$max_H = 3$

$\begin{array}{c}
\infty \\
2 \\
1 \\
0 \\
\hline
\text{head} \\
2 \\
5 \\
7 \\
10 \\
12 \\
17 \\
+\infty \\
\text{tail}
\end{array}$
Verification of Skiplists

**Skiplist shape preservation**: \(\square \text{SkipList}(h, sl, r)\)

\[
\text{SkipList}(h, sl, r) \triangleq \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0))
\]

\(\max_H = 3\)
Verification of Skiplists

- **Skiplist shape preservation**: $\square \text{SkipList}(h, sl, r)$

\[
\text{SkipList}(h, sl, r) \overset{\triangleq}{=} \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\quad \land \\
\quad r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0))
\]

\[\max_H = 3\]
Verification of Skiplists

- **Skiplist shape preservation**: \(\Box \text{SkipList}(h, sl, r)\)

\[
\text{SkipList}(h, sl, r) \triangleq \\
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \;
\wedge \\
r = p2s(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \;
\wedge \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \wedge \cdots \wedge \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null}
\]
Verification of Skiplists

**Skiplist shape preservation** : □ SkipList(h, sl, r)

\[ \text{SkipList}(h, sl, r) \equiv \]

\[ \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \]
\[ r = p2s(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \]
\[ \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H] = \text{null} \land \]
\[ a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \]
Verification of Skiplists

**Skiplist shape preservation**: \( \square \text{SkipList}(h, sl, r) \)

\[
\text{SkipList}(h, sl, r) \triangleq
\begin{align*}
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) & \land \\
r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) & \land \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} & \land \\
\cdots & \land \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H] = \text{null} & \land \\
\wedge_i \in 0 \ldots (\text{max}_H - 1) & \wedge
\end{align*}
\]

\[
\text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

\( \text{max}_H = 3 \)
Verification of Skiplists

**Skiplist shape preservation**: □ SkipList\((h, sl, r)\)

\[\text{SkipList}(h, sl, r) \triangleq \begin{align*}
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{r} = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \\
\text{a} \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \\
\land_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\end{align*}\]

\[\text{max}_H = 3\]
Verification of Skiplists

**Skiplist shape preservation**: $\square \text{SkipList}(h, sl, r)$

$\text{SkipList}(h, sl, r) \triangleq$

\[ \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \]

\[ \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H] = \text{null} \land \]

\[ a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \land \]

\[ \bigwedge_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i)) \]

$max_H = 3$

![Diagram of a skiplist with nodes and arrows indicating the structure and level of elements.](image_url)
Verification of Skiplists

➤ **Skiplist shape preservation**: □ $\text{SkipList}(h, sl, r)$

$\text{SkipList}(h, sl, r) \triangleq$

$\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0))$

$r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0))$

$\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\max_H] = \text{null}$

$a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \max_H$

$\land_{i \in 0 \ldots (\max_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))$

➤ **Program transitions**:

35: . . .

36: prev.arr[0] := x

37: . . .
Verification of Skiplists

► **Skiplist shape preservation**: □ $\text{SkipList}(h, sl, r)$

\[
\text{SkipList}(h, sl, r) \triangleq
\begin{align*}
& \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
& r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
& \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \\
& a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \\
& \bigwedge_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\end{align*}
\]

► **Program transitions**: $\text{SL}(h, sl, r)$

\[
\text{SkipList}(h, sl, r)
\]

35: . . .
36: prev.arr[0] := x
37: . . .
Verification of Skiplists

▶ Skiplist shape preservation: □ SkipList(h, sl, r)

SkipList(h, sl, r) ≡

\[
\begin{align*}
\text{ordList}(m, \text{getp}(\text{heap}, \text{sl.head}, \text{sl.tail}, 0)) & \wedge \\
r = \text{p2s}(\text{getp}(\text{heap}, \text{sl.head}, \text{sl.tail}, 0)) & \wedge \\
\text{rd}(\text{heap}, \text{sl.tail}).\text{arr}[0] = \text{null} & \wedge \cdots & \wedge \text{rd}(\text{heap}, \text{sl.tail}).\text{arr}[\text{max}_H]) = \text{null} & \wedge \\
\land_{i \in [0\ldots(\text{max}_H-1)]} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i+1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\end{align*}
\]

▶ Program transitions: SL(h, sl, r) \land \varphi_{aux}

SkipList(h, sl, r) \land 
\[
\begin{align*}
x.\text{key} & = 14 & \wedge \\
\text{prev}.\text{key} & < 14 & \wedge \\
x.\text{arr}[0].\text{key} & > 14 & \wedge \\
\text{prev}.\text{arr}[0] & = x.\text{arr}[0] & \wedge \\
x & \notin r \wedge 0 \leq 0 \leq 3
\end{align*}
\]

35: . . .
36: prev.arr[0] := x
37: . . .
Verification of Skiplists

▶ **Skiplist shape preservation**: \(\square \text{SkipList}(h, sl, r)\)

\[\text{SkipList}(h, sl, r) \triangleq \]
\[
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \land \\
a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \land \\
\bigwedge_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

▶ **Program transitions**: \(\text{SL}(h, sl, r) \land \varphi_{aux} \land \rho_{36}(V, V')\)

\[\text{SkipList}(h, sl, r) \land \\
\left(\begin{array}{c}
x.\text{key} = 14 \\
\text{prev}.\text{key} < 14 \\
x.\text{arr}[0].\text{key} > 14 \\
\text{prev}.\text{arr}[0] = x.\text{arr}[0] \\
x \notin r \land 0 \leq 0 \leq 3
\end{array}\right) \land \\
\left(\begin{array}{c}
at_{36} \\
\text{prev}'.\text{arr}[0] = x \\
at_{37} \\
h' = h \land sl = sl' \\
r' = r \cup \{x\} \land x' = x \\
\end{array}\right) \land \\
\text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

35: . . .

36: \text{prev}.\text{arr}[0] := x

37: . . .
Verification of Skiplists

**Skiplist shape preservation**: \( \square \text{SkipList}(h, sl, r) \)

\[
\text{SkipList}(h, sl, r) \equiv \begin{align*}
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \\
a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \\
\land \bigwedge_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\end{align*}
\]

**Program transitions**: \( \text{SL}(h, sl, r) \land \varphi_{\text{aux}} \land \rho_{36}(V, V') \)

\[
\text{SkipList}(h, sl, r) \land \begin{pmatrix} x.\text{key} = 14 \\ \text{prev}.\text{key} < 14 \\ x.\text{arr}[0].\text{key} > 14 \\ \text{prev}.\text{arr}[0] = x.\text{arr}[0] \\ x \notin r \land 0 \leq 0 \leq 3 \end{pmatrix} \land \begin{pmatrix} \text{at}_{36} \\ \text{prev}'.\text{arr}[0] = x \\ \text{at}'_{37} \\ h' = h \land sl = sl' \\ r' = r \cup \{x\} \land x' = x \end{pmatrix}
\]

35: . . .
36: prev.arr[0] := x
37: . . .
Verification of Skiplists

▶ Skiplist shape preservation: $\square \text{SkipList}(h, sl, r)$

$$\text{SkipList}(h, sl, r) \triangleq$$

- $\text{ordList}(m, \text{getp(heap, sl.head, sl.tail, 0)})$
- $r = \text{p2s(getp(heap, sl.head, sl.tail, 0))}$
- $\text{rd(heap, sl.tail).arr[0] = null} \land \cdots \land \text{rd(heap, sl.tail).arr[\text{max}_H]} = \text{null}$
- $a \in r \rightarrow \text{rd(heap, a).level} \leq \text{max}_H$
- $\land_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s(getp(heap, head, tail, i + 1))} \subseteq \text{p2s(getp(heap, head, tail, i))}$

▶ Program transitions: $\text{SL}(h, sl, r) \land \varphi_{aux} \land \rho_{36}(V, V')$

$$\text{SkipList}(h, sl, r) \land$$

$$\begin{pmatrix}
  x.\text{key} & = & 14 \\
  \text{prev.key} & < & 14 \\
  x.\text{arr[0].key} & > & 14 \\
  \text{prev.arr[0]} & = & x.\text{arr[0]} \\
  x & \notin r & \land 0 \leq 0 \leq 3
\end{pmatrix} \land$$

$$\begin{pmatrix}
  \text{at}_{36} \\
  \text{prev’.arr[0]} = x \\
  \text{at’}_{37} \\
  h’ = h \land sl = sl’ \\
  r’ = r \cup \{x\} \land x’ = x
\end{pmatrix} \land$$

35: . . .
36: prev.arr[0] := x
37: . . .
Verification of Skiplists

**Skiplist shape preservation:** \( \square \text{SkipList}(h, sl, r) \)

\[
\text{SkipList}(h, sl, r) \doteq \\
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
r = p2s(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \ldots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\max_H] = \text{null} \\
a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \max_H \\
\land_{i \in 0 \ldots (\max_H - 1)} p2s(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq p2s(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

**Program transitions:** \( SL(h, sl, r) \land \varphi_{aux} \land \rho_{36}(V, V') \rightarrow SL(h', sl', r') \)

\[
\text{SkipList}(h, sl, r) \land \\
\begin{cases}
  x.\text{key} = 14 \\
  \text{prev.key} < 14 \\
  x.\text{arr}[0].\text{key} > 14 \\
  \text{prev.arr}[0] = x.\text{arr}[0] \\
  x \not\in r \land 0 \leq 0 \leq 3
\end{cases} \\
\rightarrow \text{SkipList}(h', sl', r')
\]

\[
\begin{pmatrix}
  \text{at}_{36} \\
  \text{prev'.arr}[0] = x \\
  \text{at}'_{37} \\
  h' = h \land sl = sl' \\
  r' = r \cup \{x\} \land x' = x \\
\end{pmatrix}
\]

35: \ldots
36: prev.arr[0] := x
37: \ldots
Verification of Skiplists

▶ Skiplist shape preservation : □ SkipList(h, sl, r)

\[
\text{SkipList}(h, sl, r) \triangleq \begin{align*}
\text{ordList}(m, \text{getp}(heap, sl.\text{head}, sl.\text{tail}, 0)) \\
r &= \text{p2s}(\text{getp}(heap, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{rd}(heap, sl.\text{tail}).\text{arr}[0] &= \text{null} \land \cdots \land \text{rd}(heap, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \\
a \in r \rightarrow \text{rd}(heap, a).\text{level} &\leq \text{max}_H \\
\land_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(heap, head, tail, i + 1)) &\subseteq \text{p2s}(\text{getp}(heap, head, tail, i))
\end{align*}
\]

▶ Program transitions : \( SL(h, sl, r) \land \varphi_{aux} \land \rho_{36}(V, V') \rightarrow SL(h', sl', r') \)

\[
\text{SkipList}(h, sl, r) \land \\
\begin{pmatrix}
x.\text{key} &= 14 \\
\text{prev}.\text{key} &\lt 14 \\
x.\text{arr}[0].\text{key} &\gt 14 \\
\text{prev}.\text{arr}[0] &= x.\text{arr}[0] \\
x \not\in r \land 0 \leq 0 \leq 3
\end{pmatrix} \land \\
\begin{pmatrix}
at_{36} \\
\text{prev}'.\text{arr}[0] &= x \\
at'_{37} \\
h' &= h \land sl = sl' \\
r' &= r \cup \{x\} \land x' = x
\end{pmatrix} \rightarrow \text{SkipList}(h', sl', r')
\]
Verification of Skiplists

**Skiplist shape preservation:** □ $\text{SkipList}(h, sl, r)$

$\text{SkipList}(h, sl, r) \triangleq$

- $\text{ordList}(m, \text{getp}(\text{heap, sl.head, sl.tail}, 0))$
- $r = p2s(\text{getp}(\text{heap, sl.head, sl.tail}, 0))$
- $\text{rd}(\text{heap, sl.tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap, sl.tail}).\text{arr}[\max_H] = \text{null}$
- $a \in r \rightarrow \text{rd}(\text{heap, a}).\text{level} \leq \max_H$
- $\land_{i \in 0 \ldots (\max_H - 1)} \text{p2s}(\text{getp}(\text{heap, head, tail, i + 1})) \subseteq \text{p2s}(\text{getp}(\text{heap, head, tail, i}))$

**Program transitions:** $\text{SL}(h, sl, r) \land \phi_{aux} \land \rho_{36}(V, V') \rightarrow \text{SL}(h', sl', r')$

$\text{SkipList}(h, sl, r) \land$

- $x.\text{key} = 14$
- $\text{prev.key} < 14$
- $x.\text{arr}[0].\text{key} > 14$
- $\text{prev.arr}[0] = x.\text{arr}[0]$
- $x \notin r \land 0 \leq 0 \leq 3$

$\land$

- $\text{at}_{36}$
- $\text{prev'.arr}[0] = x$
- $\text{at}'_{37}$
- $h' = h \land sl = sl'$
- $r' = r \cup \{x\} \land x' = x$

$\rightarrow \text{SkipList}(h', sl', r')$

**reason about**

ordered values + notion of ordered list
Verification of Skiplists

**Skiplist shape preservation**: \( \square \text{SkipList}(h, sl, r) \)

\[
\text{SkipList}(h, sl, r) \triangleq \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H] = \text{null} \land \\
\land_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

**Program transitions**: \( \text{SL}(h, sl, r) \land \varphi_{\text{aux}} \land \rho_{36}(V, V') \rightarrow \text{SL}(h', sl', r') \)

\[
\text{SkipList}(h, sl, r) \land \\
\begin{pmatrix}
    x.\text{key} = 14 \\
    \text{prev.key} < 14 \\
    x.\text{arr}[0].\text{key} > 14 \\
    \text{prev.arr}[0] = x.\text{arr}[0] \\
    x \notin r \land 0 \leq 0 \leq 3
\end{pmatrix} \land \\
\begin{pmatrix}
    \text{at}_{36} \\
    \text{prev'.arr}[0] = x \\
    \text{at}'_{37} \\
    h' = h \land sl = sl' \\
    r' = r \cup \{x\} \land x' = x
\end{pmatrix} \rightarrow \text{SkipList}(h', sl', r')
\]
Verification of Skiplists

**Skiplist shape preservation**: □\textit{SkipList}(h, sl, r)

\[
\text{SkipList}(h, sl, r) \triangleq \\
\text{ordList}(m, \text{getp}(\text{heap, sl.head, sl.tail}, 0)) \quad \wedge \\
r = \text{p2s}(\text{getp}(\text{heap, sl.head, sl.tail}, 0)) \quad \wedge \\
\text{rd}(\text{heap, sl.tail})[\text{arr}[0] = \text{null} \wedge \cdots \wedge \text{rd}(\text{heap, sl.tail})[\text{arr}[\text{max}_H]] = \text{null} \quad \wedge \\
a \in r \rightarrow \text{rd}(\text{heap, a}).\text{level} \leq \text{max}_H \\
\wedge_{i \in 0 \ldots (\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap, head, tail, i + 1})) \subseteq \text{p2s}(\text{getp}(\text{heap, head, tail, i}))
\]

**Program transitions**: \textit{SL}(h, sl, r) \land \varphi_{aux} \land \rho_{36}(V, V') \rightarrow \text{SL}(h', sl', r')

\[
\text{SkipList}(h, sl, r) \land \\
x.\text{key} = 14 \quad \wedge \\
\text{prev.key} < 14 \quad \wedge \\
x.\text{arr}[0].\text{key} > 14 \quad \wedge \\
\text{prev.}\text{arr}[0] = x.\text{arr}[0] \quad \wedge \\
x \notin r \land 0 \leq 0 \leq 3 \\
\]

\[
\quad \wedge \\
\quad \wedge \\
\quad \wedge \\
\quad \wedge \\
\]

\[
\rightarrow \text{SkipList}(h', sl', r') \\
\]

reason about arrays
Verification of Skiplists

- **Skiplist shape preservation**: □SkipList(h, sl, r)

\[ \text{SkipList}(h, sl, r) \triangleq \]
\[ \text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \]
\[ r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \land \]
\[ \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \land \]
\[ \land_{i \in 0\ldots(\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i)) \]

- **Program transitions**: \( SL(h, sl, r) \land \varphi_{\text{aux}} \land \rho_{36}(V, V') \rightarrow SL(h', sl', r') \)

\[ \text{SkipList}(h, sl, r) \land \]
\[ \begin{align*}
  x.\text{key} & = 14 \land \\
  \text{prev}.\text{key} & < 14 \land \\
  x.\text{arr}[0].\text{key} & > 14 \land \\
  \text{prev}.\text{arr}[0] & = x.\text{arr}[0] \land \\
  x \notin r \land 0 \leq 0 \leq 3
\end{align*} \]
\[ \land \begin{align*}
  \text{at}_{36} & \land \\
  \text{prev}'.\text{arr}[0] & = x \land \\
  \text{at}_{37}' & \land \\
  h' & = h \land sl = sl' \land \\
  r' & = r \cup \{x\} \land x' = x \ldots
\end{align*} \]
\[ \rightarrow \text{SkipList}(h', sl', r') \]

reason about

regions (sets)
Verification of Skiplists

- **Skiplist shape preservation**: $\square \text{SkipList}(h, sl, r)$

\[
\text{SkipList}(h, sl, r) \doteq \\
\text{ordList}(m, \text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
r = \text{p2s}(\text{getp}(\text{heap}, sl.\text{head}, sl.\text{tail}, 0)) \\
\text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[0] = \text{null} \land \cdots \land \text{rd}(\text{heap}, sl.\text{tail}).\text{arr}[\text{max}_H]) = \text{null} \\
a \in r \rightarrow \text{rd}(\text{heap}, a).\text{level} \leq \text{max}_H \\
\land \sum_{i=0}^{(\text{max}_H - 1)} \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i + 1)) \subseteq \text{p2s}(\text{getp}(\text{heap}, \text{head}, \text{tail}, i))
\]

- **Program transitions**: $\text{SL}(h, sl, r) \land \varphi_{\text{aux}} \land \rho_{36}(V, V') \rightarrow \text{SL}(h', sl', r')$

\[
\text{SkipList}(h, sl, r) \land \\
\begin{cases}
  x.\text{key} = 14 & \land \\
  \text{prev}.\text{key} < 14 & \land \\
  x.\text{arr}[0].\text{key} > 14 & \land \\
  \text{prev}.\text{arr}[0] = x.\text{arr}[0] & \land \\
  x \notin r \land 0 \leq 0 \leq 3 & \\
\end{cases} \\
\land \\
\begin{cases}
  \text{at}_{36} & \land \\
  \text{prev}'.\text{arr}[0] = x & \land \\
  \text{at}'_{37} & \land \\
  h' = h \land sl = sl' & \land \\
  r' = r \cup \{x\} \land x' = x & \land \\
\end{cases} \rightarrow \text{SkipList}(h', sl', r')
\]

reason about

memory, cells
Our Contribution

- **TSL**, a theory for skiplists of *arbitrary length and height*

- We show TSL *decidable*...

- ...by reducing **TSL satisfiability** to **TSL<sub>K</sub> satisfiability**.
TSL: A Theory for Skiplists of Arbitrary Height
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $\text{TSL}_K$, is a union of other theories
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \sum_{addr} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSLₖ, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \]
TSL: A Theory for SkipLists of Arbitrary Height

- TSL, like TSL_κ, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}}$$
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $TSL_K$, is a union of other theories

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}}$$
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \]
TSL: A Theory for SkipLists of Arbitrary Height

▶ TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}}$$
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $TSL_K$, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \]
TSL: A Theory for Skiplists of Arbitrary Height

TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}} \]

path = a non-repeating sequence of addresses

\[ [a_1, a_2, a_3] \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[
\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}}
\]

\[
\text{append}([a_1, a_2], [a_3], [a_1, a_2, a_3])
\]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}}$$

$$\text{reach}(a_0, a_3, 1, [a_0, a_2])$$
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{addr} \cup \Sigma_{elem} \cup \Sigma_{ord} \cup \Sigma_{cell} \cup \Sigma_{mem} \cup \Sigma_{set} \cup \Sigma_{reachability} \cup \Sigma_{bridge} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $TSL_K$, is a union of other theories

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

$$\text{path2set}([a_2, a_3]) = \{a_2, a_3\}$$
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL\(_K\), is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}} \cup \Sigma_{\text{bridge}} \]

\[ \text{addr2set}(a_0, 1) = \{ a_0, a_2, a_3 \} \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[
\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}} \cup \Sigma_{\text{bridge}}
\]

\[
\text{getp}(a_0, a_2, 0) = [a_0, a_1]
\]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[ \Sigma_{addr} \cup \Sigma_{elem} \cup \Sigma_{ord} \cup \Sigma_{cell} \cup \Sigma_{mem} \cup \Sigma_{set} \cup \Sigma_{reachability} \cup \Sigma_{bridge} \]

\[ ordList([a_0, a_1, a_2, a_3]) \]
TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL$_K$, is a union of other theories

\[ \Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}} \cup \Sigma_{\text{bridge}} \]
Decision Procedure for TSL
Decision Procedure for TSL

- Let $\varphi$ be a normalized TSL formula
Decision Procedure for TSL

Let $\varphi$ be a normalized TSL formula
Decision Procedure for TSL

- Let $\varphi$ be a normalized TSL formula
Let $\varphi$ be a normalized TSL formula
Decision Procedure for TSL

Let $\varphi$ be a normalized TSL formula
Decision Procedure for TSL

Let $\varphi$ be a normalized TSL formula $\varphi$. 

Decision Procedure for TSL

Let $\varphi$ be a normalized TSL formula

1. guess arrangement $\alpha$
Decision Procedure for TSL

- Let $\varphi$ be a normalized TSL formula

1. guess arrangement $\alpha$

$\varphi^{PA} \land \alpha$  $\varphi^{NC} \land \alpha$
Decision Procedure for TSL

- Let \( \varphi \) be a normalized TSL formula

1. guess arrangement \( \alpha \)

\[
\varphi^{PA} \land \alpha \quad \leftarrow \quad \varphi \\
\varphi^{NC} \land \alpha \\
\]

2. check satisfiability
Decision Procedure for TSL

- Let $\varphi$ be a normalized TSL formula

1. guess arrangement $\alpha$

2. check satisfiability

3. check satisfiability
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

\[
\varphi \quad \varphi^{PA} \land \alpha \quad \varphi^{NC} \land \alpha
\]
Let $\varphi$ be a normalized TSL formula

\[ \varphi \]

\[ \varphi^{PA} \land \alpha \]  
\[ \varphi^{NC} \land \alpha \]

Theorem:

$\varphi$: TSL formula is satisfiable

iff

$(\varphi^{PA} \land \alpha)$ is satisfiable and $(\varphi^{NC} \land \alpha)$ is satisfiable
Let $\varphi$ be a normalized TSL formula.

Presburguer Arithmetic
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

$\varphi$

$\varphi^{PA} \land \alpha$ ← $\varphi$

$\varphi^{NC} \land \alpha$

- Gapless model: we stay only with interesting levels
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

\[ \varphi \]

$\varphi^{PA} \land \alpha$ \quad $\varphi^{NC} \land \alpha$

$V_{level}(\varphi^{NC} \land \alpha) = \{l_1, l_3\}$

- Gapless model: we stay only with interesting levels

\[ \begin{array}{c}
\infty \\
-2 \\
-5 \\
-7 \\
-10 \\
-12 \\
-14 \\
-17 \\
+2 \\
+5 \\
+7 \\
+10 \\
+12 \\
+14 \\
+17 \\
+\infty
\end{array} \]

head \quad tail
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

$$\varphi$$

$$\varphi^{PA} \land \alpha \quad \leftarrow \quad V_{\text{level}}(\varphi^{NC} \land \alpha) = \{l_1, l_3\}$$

- Gapless model: we stay only with interesting levels

head

$-\infty$ 2 5 7 10 12 14 17 $+\infty$

tail
Let $\varphi$ be a normalized TSL formula

$\varphi^{PA} \land \alpha$ \hspace{1cm} $\varphi^{NC} \land \alpha$

We reduce the formula to $\varphi^{NC} \land \alpha^\perp$:

**Gapless model**: we stay only with **interesting levels**
Let $\varphi$ be a normalized TSL formula.

We reduce the formula to $\varphi^{NC} \land \alpha$:

\[ \Box \varphi^{PA} \land \alpha \quad \text{and} \quad \varphi^{NC} \land \alpha \]

Theorem:

$\varphi : TSL$ formula without constant levels is satisfiable

iff

$\varphi^\neg : TSL_K$ is satisfiable
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\varphi$

$\varphi^{PA} \land \alpha$

$\varphi^{NC} \land \alpha$

Reduction

TSL $\longrightarrow$ TSL$_K$
Let $\varphi$ be a normalized TSL formula.

\[ \varphi \]

- Reduction

\[ \rightarrow \]

\[ \varphi^{PA} \land \alpha \] \hspace{1cm} \varphi^{NC} \land \alpha \]

\[ \varphi^{PA} \land \alpha \] \hspace{1cm} \varphi^{NC} \land \alpha \]

- Reduction

\[ \text{TSL} \rightarrow \text{TSL}_K \]

\[ \left( c = \text{mkcell}(e, k, A, l) \right)^{-} \]

\[ c = (e, k, v_A[0], \ldots, v_A[K-1]) \]

\[ v_A[l] = A(l) \]
Let $\varphi$ be a normalized TSL formula

$\varphi^{PA} \land \alpha$ \hspace{1cm} $\varphi^{NC} \land \alpha$

**Reduction**

\[
\begin{align*}
\lceil c = mkcell(e, k, A, l) \rceil &\quad \Rightarrow \quad c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]}) \\
\lceil a = A[l] \rceil &\quad \land \quad l = i \rightarrow a = v_{A[i]} \quad \text{for } i = 0, \ldots, K-1
\end{align*}
\]
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

\[ \varphi \]

\[ \varphi^{PA} \land \alpha \]  
\[ \varphi^{NC} \land \alpha \]

Reduction

\[ \text{TSL} \quad \longrightarrow \quad \text{TSL}_K \]

\[ \begin{align*}
\Gamma c = \text{mkcell}(e, k, A, l) & \quad c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]}) \\
\Gamma a = A[l] & \quad \bigwedge_{i=0\ldots K-1} l = i \rightarrow a = v_{A[i]} \\
\Gamma B = A\{l \leftarrow a\} & \quad \left( \bigwedge_{i=0\ldots K-1} l = i \rightarrow a = v_{B[i]} \right) \land \\
& \quad \left( \bigwedge_{j=0\ldots K-1} l \neq j \rightarrow v_{B[j]} = v_{A[j]} \right)
\end{align*} \]
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

\[
\varphi^P \land \alpha \quad \text{and} \quad \varphi^{NC} \land \alpha
\]

$\exists_{TSL} \mathcal{M}$ model

$\exists_{TSLk} \mathcal{M}''$ model
Let $\varphi$ be a normalized TSL formula.
Decision Procedure for TSL: Correctness

- Let \( \varphi \) be a normalized TSL formula

\[ \varphi^{PA} \land \alpha \]

\[ \varphi^{NC} \land \alpha \]

\[ \exists_{\text{TSL}} \mathcal{M} \text{ model} \]

\[ \exists_{\text{TSL}(\text{gapless})} \mathcal{M}' \text{ model} \]

\[ \exists_{\text{TSL}_k} \mathcal{M}'' \text{ model} \]

\[ \neg \varphi^{NC} \land \alpha \neg \]

Not that easy!
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$$B = A\{l_1 \leftarrow a\}$$

$\exists_{TSL} M$ model

$\exists_{TSL\text{(gapless)}} M'$ model

$\exists_{TSL_k} M''$ model

$$\varphi^{PA} \land \alpha$$

$$\varphi^{NC} \land \alpha$$

$$\varphi$$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\varphi_{PA} \land \alpha$ ☐

$\varphi_{NC} \land \alpha$ ☐

$B = A\{l_1 \leftarrow a\}$

$\exists_{TSL} M$ model

$\exists_{TSL\text{(gapless)}} M'$ model

$\exists_{TSL \text{k}} M''$ model

$\Gamma \varphi_{NC} \land \alpha$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\exists_{\text{TSL}} \mathcal{M}$ model

$\exists_{\text{TSL}_{\text{gapless}}} \mathcal{M}'$ model

$\exists_{\text{TSL}_K} \mathcal{M}''$ model

$B = A\{l_1 \leftarrow a\}$
Let $\varphi$ be a normalized TSL formula

$$
\varphi \; P.A \land \alpha
$$

$$
\exists_{TSL} M \; \text{model}
$$

$$
\exists_{TSL_{(gapless)}} M' \; \text{model}
$$

$$
\exists_{TSL_k} M'' \; \text{model}
$$

$$
B = A\{l_1 \leftarrow a\}
$$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\varphi = \varphi^{PA} \land \alpha$

$\exists_{TSL} M$ model

$\exists_{TSL(gapless)} M'$ model

$A \xrightarrow{l_2} B = A \{l_1 \leftarrow a\}$

$\exists_{TSL_k} M''$ model

$\Gamma \varphi^{NC} \land \alpha$
Let $\varphi$ be a normalized TSL formula.

$\varphi$ is a formula in TSL.

$\varphi^{PA} \land \alpha$

$\exists_{TSL} \mathcal{M}$ model

$\exists_{TSL\,(gapless)} \mathcal{M}'$ model

$\exists_{TSL_k} \mathcal{M}''$ model

$\Gamma \varphi^{NC} \land \alpha$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\varphi^P A \land \alpha$  \rightarrow  $\varphi^{NC} \land \alpha$

$\exists_{TSL} M$ model

$\exists_{TSL_{(gapless)}} M'$ model

$\exists_{TSL_k} M''$ model

$\Gamma \varphi^{NC} \land \alpha$
Let $\varphi$ be a normalized TSL formula

$\varphi^{PA} \land \alpha$

$B = A\{l_1 \leftarrow a\}$

Add $l_{new} = l_1 + 1$

$\exists_{TSL} M$ model

$\exists_{TSL(gapless)} M'$ model

$\exists_{TSL_k} M''$ model

$\Gamma \varphi^{NC} \land \alpha \vdash$
Decision Procedure for TSL: Correctness

- Let $\varphi$ be a normalized TSL formula

$\varphi^{PA}$ \land \alpha \quad \checkmark \quad \varphi^{NC}$ \land \alpha

$B = A\{l_1 \leftarrow a\}$
Add $l_{new} = l_1 + 1$

$\exists_{TSL} M$ model
$
\exists_{TSL\text{(gapless)}} M'$ model
$
\exists_{TSL_K} M''$ model
$
\models \varphi^{NC} \land \alpha$

$A$

$B$

Add $l_{new} = l_1 + 1$
Let $\varphi$ be a normalized TSL formula.

$\varphi$

$\varphi^{PA} \land \alpha$

$\varphi^{NC} \land \alpha$

$\exists_{TSL} \mathcal{M}$ model

$\exists_{TSL\text{(gapless)}} \mathcal{M}'$ model

$\exists_{TSL_k} \mathcal{M}''$ model

$\Gamma \varphi^{NC} \land \alpha$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula.

$\varphi^{PA} \land \alpha \iff \varphi^{NC} \land \alpha$

$\exists_{TSL} M \text{ model}$

$\exists_{TSL_{(gapless)}} M' \text{ model}$

$\exists_{TSL_{k}} M'' \text{ model}$

$\Gamma \varphi^{NC} \land \alpha \downarrow$
Decision Procedure for TSL: Correctness

Let $\varphi$ be a normalized TSL formula

$\varphi$

$\varphi^{PA} \land \alpha$

Presburguer Arithmetic

Reduce TSL satisfiability to

$\exists_{TSL} M$ model

$\exists_{TSL(gapless)} M'$ model

$\exists_{TSL_K} M''$ model

$\Gamma \varphi^{NC} \land \alpha$
Conclusions

▶ We defined **TSL**, a theory for skiplists of arbitrary height

▶ We proved TSL **decidable**...

▶ ... by reducing to **Presburguer Arithmetic** and **TSL₀**

▶ **TSL₀** can reason about memory, cells, pointers, regions, reachability, ordered lists and sublists

▶ **Current and future** work:
   have implementation of DP for **TSL₀**
   building DP for TSL
   thinking on DP for concurrent skiplists

▶ Many possible **collaborations**:  
   DPs as combination, SMTs, implementation