Verification: Overview

CS156: Calculus of Computation

Zohar Manna Aaron R. Bradley

Outline

⇒ 0. Motivation

1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
   (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures
   (a) Overview
   (b) Theories

BinarySearch: Motivation

```
bool BinarySearch(int[] a, int e) {
  return BSearch(a, 0, |a| − 1, e);
}

bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BSearch(a, m + 1, u, e);
    else return BSearch(a, ℓ, m − 1, e);
  }
}
```

Does BinarySearch return true iff sorted array a contains e?

Specification

@pre: **precondition** of function.

- What must be true when function is called?
- Assertion over function parameters.

@post: **postcondition** of function.

- If precondition is true, then postcondition is true on exit.
- Assertion over function parameters and return value, rv.

BinarySearch: Specification

If array a is sorted and BinarySearch terminates,
then it returns **true** iff a contains e.

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$ precondition

ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$ postcondition

```
bool BinarySearch(int[] a, int e) {
  return BSearch(a, 0, |a| − 1, e);
}
```

BinarySearch: Specification

If array a is sorted and BSearch terminates,
then it returns **true** iff a contains e
between positions ℓ and u.

m_0: @pre $0 \leq \ell \ \wedge \ u < |a| \ \wedge$ sorted$(a, 0, |a| - 1)$ precondition
m_f: @post rv $\leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]$ postcondition

```
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BSearch(a, m + 1, u, e);
    else return BSearch(a, ℓ, m − 1, e);
  }
}
```

BubbleSort: Motivation

```
int[] BubbleSort(int[] a) {
  int i, j, t;
  for (i := |a| − 1; i > 0; i := i − 1) {
    for (j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Does BubbleSort return a sorted array?

BubbleSort: Specification

If BubbleSort terminates, then it returns a sorted array.

```
ℓ0: @pre |a| ≥ 0                                          precondition
ℓf: @post sorted(rv, 0, |a| − 1)                          postcondition
int[] BubbleSort(int[] a) {
  int i, j, t;
  for (i := |a| − 1; i > 0; i := i − 1) {
    for (j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Correctness

Program: Sequence of instructions.

Specification: Formal statement of what program should do.

> **Goal**: Prove program correct; *i.e.*, that it meets specification.

Partial Correctness: Prove

> If the precondition is true initially
> and the function exits,
> then the postcondition holds on exit.

Total Correctness: Also prove **termination**

> If the precondition is true initially
> then the function eventually exits.

Partial Correctness

Strategy: Analyze programs as mathematical objects.

- Generate and prove **verification conditions** (VCs) from **program** and **specification**.
- Prove that all VCs are **valid**.

That is, translate the verification problem
to the problem of proving validity of VCs.

Partial Correctness

Method:

- Annotate functions with **preconditions** and **postconditions**.
- Annotate loops with **loop invariants**.
- Precondition is first assertion.
  Postcondition is last assertion.
- Generate and prove **verification conditions**
  - for each path from one assertion φ to the next ψ:
    if φ is true at the start of the path, then ψ is true at the end
  - for each function call:
    preconditions are true in the calling context

Based on mathematical induction.

Outline

0. Motivation
⇒ 1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
   (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures
   (a) Overview
   (b) Theories

BinarySearch: Transition System

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$ precondition
ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$ postcondition

```
bool BinarySearch(int[] a, int e) {
  return BSearch(a, 0, |a| − 1, e);
}
```

m_0: @pre $0 \leq \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$ precondition
m_f: @post rv $\leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]$ postcondition

```
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BSearch(a, m + 1, u, e);
    else return BSearch(a, ℓ, m − 1, e);
  }
}
```

BinarySearch: Transition System and Path

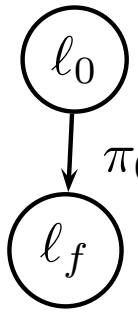
Input: sorted integer array a

Output: **true** iff $(\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$

ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$

```
bool BinarySearch(int[] a, int e) {
  return BSearch(a, 0, |a| − 1, e);                    ... π0
}
```



ρ_{π_0} : $\quad \mathsf{rv}' = \text{BSearch}(a, 0, |a| - 1, e) \ \wedge \ \mathsf{pres}(\ldots)$

$\mathsf{pres}(V)$: preserve values of $V \subseteq \mathcal{V}$

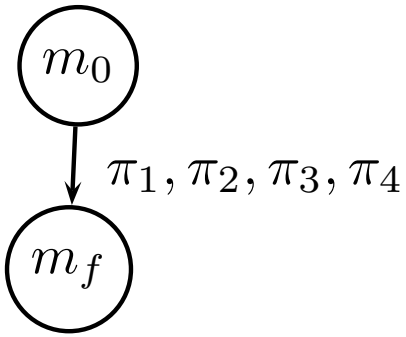
$\mathsf{pres}(\ldots)$: preserve values of unmentioned variables of $\mathcal{V}$

BinarySearch: Transition System

m_0: @pre $0 \leq \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$

m_f: @post rv $\leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]$

```
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;                                  ... π1
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;                              ... π2
    else if (a[m] < e) return BSearch(a, m + 1, u, e);      ... π3
    else return BSearch(a, ℓ, m − 1, e);                    ... π4
  }
}
```



BinarySearch: Transition System and Path

m_0: @pre $0 \le \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a|-1)$

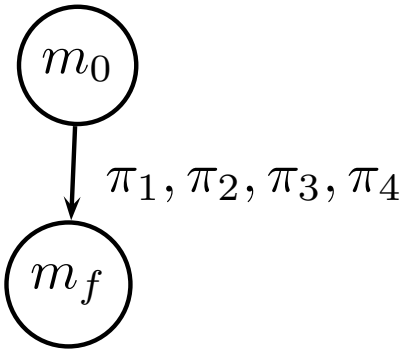
m_f: @post rv $\ \leftrightarrow \ (\exists i)[\ell \le i \le u \ \wedge \ a[i] = e]$

```
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;                                   ... π2
    else if (a[m] < e) return BSearch(a, m + 1, u, e);
    else return BSearch(a, ℓ, m − 1, e);
  }
}
```

$$\rho_{\pi_2} : \ \ell \le u \ \wedge \ m' = \tfrac{\ell+u}{2} \ \wedge \ a\left[\tfrac{\ell+u}{2}\right] = e \ \wedge \ \mathsf{rv}' = \mathbf{true} \ \wedge \ \mathsf{pres}(\ldots)$$

$\uparrow$ integer division

BSearch: Paths



$$\rho_{\pi_1} : \ell > u \ \wedge \ \mathsf{rv}' = \mathbf{false} \ \wedge \ \mathsf{pres}(\ldots)$$

$$\rho_{\pi_2} : \begin{array}{l} \ell \leq u \ \wedge \ m' = \frac{\ell+u}{2} \ \wedge \ a[\frac{\ell+u}{2}] = e \\ \wedge \ \mathsf{rv}' = \mathbf{true} \ \wedge \mathsf{pres}(\ldots) \end{array}$$

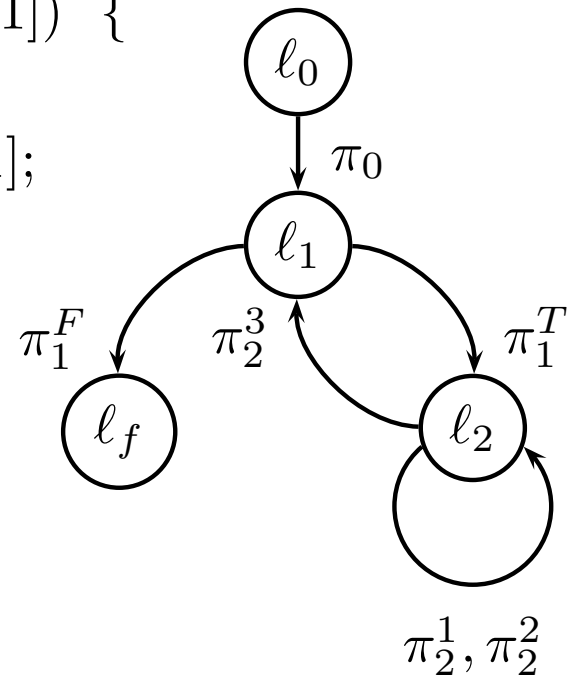
$$\rho_{\pi_3} : \begin{array}{l} \ell \leq u \ \wedge \ m' = \frac{\ell+u}{2} \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] < e \\ \wedge \ \mathsf{rv}' = \textsc{BSearch}(a, \frac{\ell+u}{2} + 1, u, e) \ \wedge \ \mathsf{pres}(\ldots) \end{array}$$

$$\rho_{\pi_4} : \begin{array}{l} \ell \leq u \ \wedge \ m' = \frac{\ell+u}{2} \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] \geq e \\ \wedge \ \mathsf{rv}' = \textsc{BSearch}(a, \ell, \frac{\ell+u}{2} - 1, e) \ \wedge \ \mathsf{pres}(\ldots) \end{array}$$

BubbleSort: Transition System

ℓ_0: @pre $|a| \geq 0$
ℓ_f: @post sorted(rv, $0, |a| - 1$)

```
int[] BubbleSort(int[] a) {
  int i, j, t;
  for (i := |a| − 1; [ℓ1]: i > 0; i := i − 1) {
    for (j := 0; [ℓ2]: j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```



π_0 : $\langle \ell_0, \ell_1, \rho_{\pi_0} \rangle$
π_1^T : $\langle \ell_1, \ell_2, \rho_{\pi_1^T} \rangle$
π_1^F : $\langle \ell_1, \ell_f, \rho_{\pi_1^F} \rangle$
π_2^1 : $\langle \ell_2, \ell_2, \rho_{\pi_2^1} \rangle$
π_2^2 : $\langle \ell_2, \ell_2, \rho_{\pi_2^2} \rangle$
π_2^3 : $\langle \ell_2, \ell_1, \rho_{\pi_2^3} \rangle$

BubbleSort: Paths

ρ_{π_0} : $\quad i' = |a| - 1 \ \wedge \ \mathsf{pres}(\ldots)$

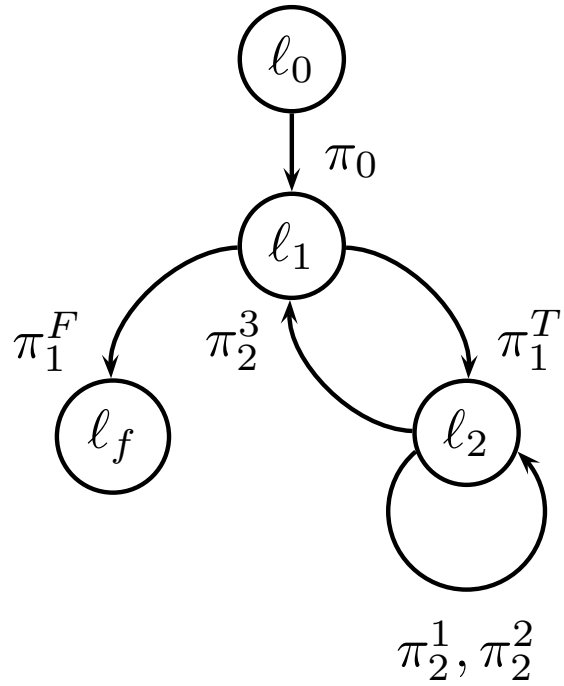
$\rho_{\pi_1^T}$: $\quad i > 0 \ \wedge \ j' = 0 \ \wedge \ \mathsf{pres}(\ldots)$

$\rho_{\pi_1^F}$: $\quad i \leq 0 \ \wedge \ \mathsf{rv}' = a \ \wedge \ \mathsf{pres}(\ldots)$

$\rho_{\pi_2^1}$: $\quad j < i \ \wedge \ a[j] > a[j+1] \ \wedge \ t' = a[j]$
$\wedge \ a' = a[j : a[j+1]][j+1 : a[j]] \ \wedge \ j' = j + 1 \ \wedge \ \mathsf{pres}(\ldots)$

$\rho_{\pi_2^2}$: $\quad j < i \ \wedge \ a[j] \leq a[j+1] \ \wedge \ j' = j + 1 \ \wedge \ \mathsf{pres}(\ldots)$

$\rho_{\pi_2^3}$: $\quad j \geq i \ \wedge \ i' = i - 1 \ \wedge \ \mathsf{pres}(\ldots)$



$\mathsf{pres}(V)$: preserve values of $V \subseteq \mathcal{V}$

$\mathsf{pres}(\ldots)$: preserve values of unmentioned variables of $\mathcal{V}$

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness

⇒ (a) Annotations

(b) Function Calls

(c) Runtime Assertions

3. Total Correctness (Termination)
4. Decision Procedures

(a) Overview

(b) Theories

BubbleSort: Annotations

ℓ_0: @pre $|a| \geq 0$ — input specification

ℓ_f: @post sorted(rv, $0, |a| - 1$) — output specification

```
int[] BubbleSort(int[] a) {
  int i, j, t;
```

loop assertions

```
  for ℓ1: @ [ −1 ≤ i < |a| ∧ |a| = |a|_0
              ∧ partitioned(a, 0, i, i + 1, |a| − 1) ∧ sorted(a, i, |a| − 1) ]
  (i := |a| − 1; i > 0; i := i − 1)
      for ℓ2: @ [ 1 ≤ i < |a| ∧ 0 ≤ j ≤ i ∧ |a| = |a|_0
                  ∧ partitioned(a, 0, i, i + 1, |a| − 1)
                  ∧ partitioned(a, 0, j − 1, j, j) ∧ sorted(a, i, |a| − 1) ]
      (j := 0; j < i; j := j + 1)
        if (a[j] > a[j + 1]) {
          t := a[j]; a[j] := a[j + 1]; a[j + 1] := t;
        }
  return a;
}
```

BubbleSort: Annotations

Predicates:

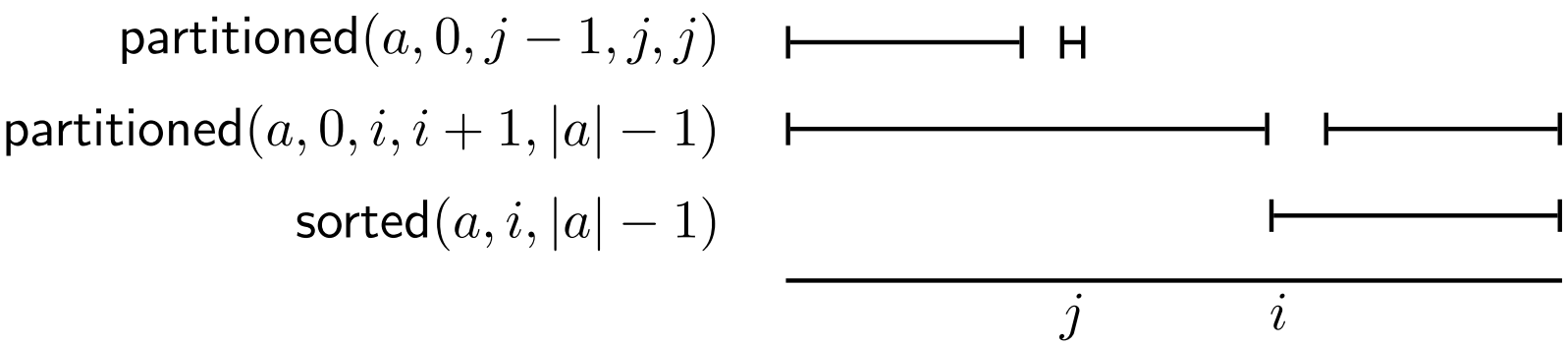
- $\mathsf{sorted}(a, \ell, u)$: array a is sorted in range $[\ell, u]$

$$(\forall i, j)[\ell \leq i \leq j \leq u \ \rightarrow \ a[i] \leq a[j]]$$

- $\mathsf{partitioned}(a, \ell_1, u_1, \ell_2, u_2)$:

$$(\forall i, j)[\ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \ \rightarrow \ a[i] \leq a[j]]$$

At the top of the inner loop:



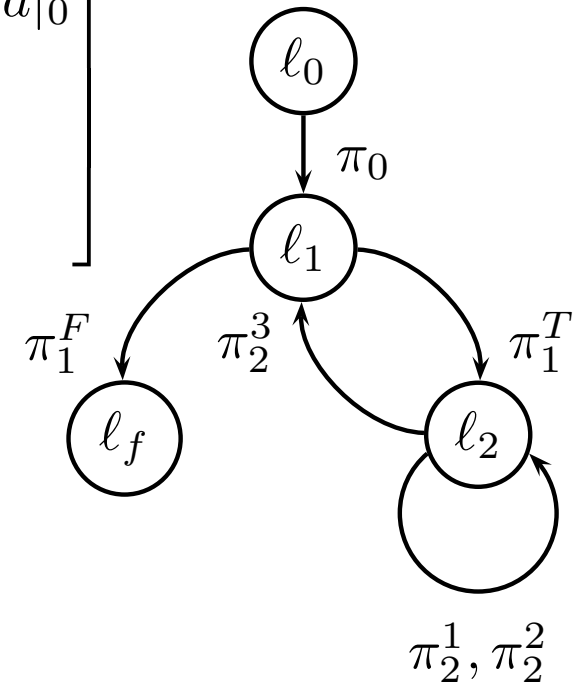
BubbleSort: Assertion Map

$$\mu(\ell_0) = |a| \geq 0$$

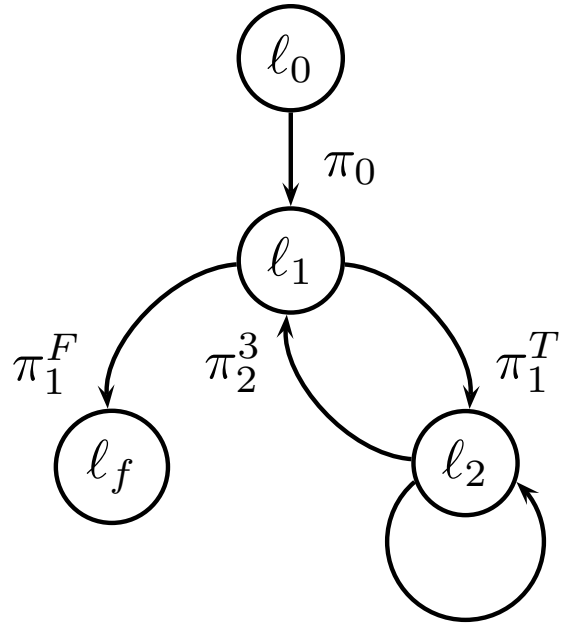
$$\mu(\ell_1) = \begin{bmatrix} -1 \leq i < |a| \ \wedge \ |a| = |a|_0 \\ \wedge \ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \ \wedge \ \mathsf{sorted}(a, i, |a|-1) \end{bmatrix}$$

$$\mu(\ell_2) = \begin{bmatrix} 1 \leq i < |a| \ \wedge \ 0 \leq j \leq i \ \wedge \ |a| = |a|_0 \\ \wedge \ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \mathsf{partitioned}(a, 0, j-1, j, j) \\ \wedge \ \mathsf{sorted}(a, i, |a|-1) \end{bmatrix}$$

$$\mu(\ell_f) = \mathsf{sorted}(\mathsf{rv}, 0, |a|-1)$$



BubbleSort: Partial Correctness

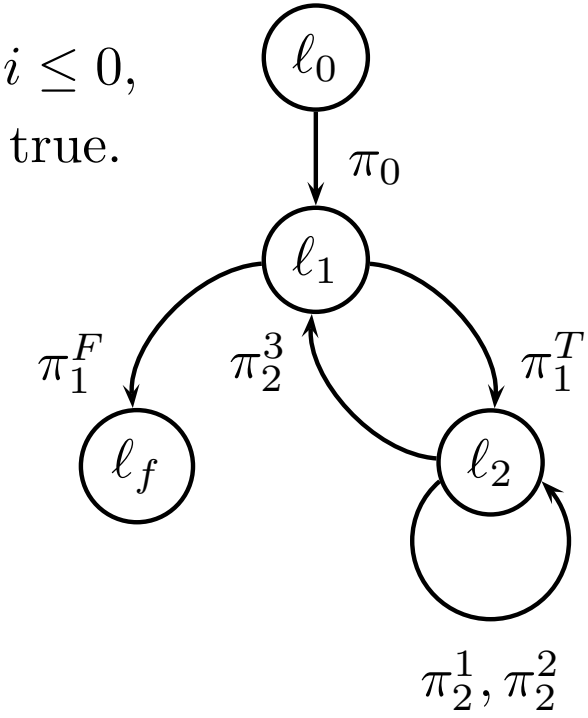


Prove that

- π_0: if precondition $\mu(\ell_0)$ is true and $i' = |a| - 1$, then outer loop assertion $\mu(\ell_1)'$ is true;
- π_1^T: if outer loop assertion $\mu(\ell_1)$ is true and $i > 0 \ \wedge \ j' = 0$, then inner loop assertion $\mu(\ell_2)'$ is true;
- π_2^1: if inner loop assertion $\mu(\ell_2)$ is true, $j < i \ \wedge \ a[j] \leq a[j+1]$, and j is incremented, then inner loop assertion $\mu(\ell_2)'$ is true;

BubbleSort: Partial Correctness

- π_2^2: if inner loop assertion $\mu(\ell_2)$ is true, $j < i \ \wedge \ a[j] > a[j+1]$, $a[j]$ and $a[j+1]$ are swapped, and j is incremented, then inner loop assertion $\mu(\ell_2)'$ is true;
- π_2^3: if inner loop assertion $\mu(\ell_2)$ is true, $j \geq i$, and i is decremented, then outer loop assertion $\mu(\ell_1)'$ is true;
- π_1^F: if outer loop assertion $\mu(\ell_1)$ is true, $i \leq 0$, and $\mathsf{rv}' = a$, then postcondition $\mu(\ell_f)'$ is true.



Invariant Map

An assertion map μ is an **invariant map** if

for each path $\pi : \langle \ell, m, \rho_\pi \rangle$,

if $\mu(\ell)$ holds now,

then taking π makes $\mu(m)$ hold next.

That is, if for each path π the **verification condition** (VC)

$$\underbrace{(\forall *)[\mu(\ell) \wedge \rho_\pi \ \rightarrow \ \mu(m)']}_{\{\mu(\ell)\}\rho_\pi\{\mu(m)\}}$$

is valid.

Note the similarity to mathematical induction.

BubbleSort: Verification Condition

$\{\mu(\ell_2)\}\rho_{\pi_2^2}\{\mu(\ell_2)\}$:

$$
(\forall *)\ \left[\begin{array}{l}
1 \le i < |a| \ \wedge\ 0 \le j \le i \ \wedge\ |a| = |a|_0 \\
\wedge\ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \\
\underbrace{\wedge\ \mathsf{partitioned}(a, 0, j-1, j, j) \ \wedge\ \mathsf{sorted}(a, i, |a|-1)}_{\mu(\ell_2)} \\
\wedge\ \underbrace{j < i \ \wedge\ a[j] > a[j+1]}_{\text{guard of } \rho_{\pi_2^2}} \\
\quad 1 \le i < |a| \ \wedge\ 0 \le j+1 \le i \ \wedge\ |a| = |a|_0 \\
\quad \wedge\ \mathsf{partitioned}(a[j : a[j+1]][j+1 : a[j]], 0, i, i+1, |a|-1) \\
\rightarrow\ \wedge\ \mathsf{partitioned}(a[j : a[j+1]][j+1 : a[j]], 0, j, j+1, j+1) \\
\quad \underbrace{\wedge\ \mathsf{sorted}(a[j : a[j+1]][j+1 : a[j]], i, |a|-1)}_{\mu(\ell_2)\{j \mapsto j+1,\ a \mapsto a[j:a[j+1]][j+1:a[j]]\}}
\end{array}\right]
$$

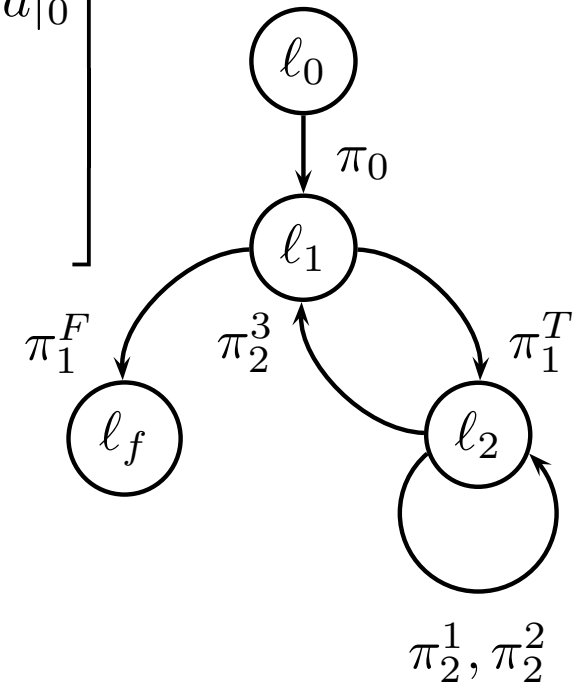
BubbleSort: Invariant Map

$$\mu(\ell_0) = |a| \geq 0$$

$$\mu(\ell_1) = \begin{bmatrix} -1 \leq i < |a| \ \wedge \ |a| = |a|_0 \\ \wedge \ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \ \wedge \ \mathsf{sorted}(a, i, |a|-1) \end{bmatrix}$$

$$\mu(\ell_2) = \begin{bmatrix} 1 \leq i < |a| \ \wedge \ 0 \leq j \leq i \ \wedge \ |a| = |a|_0 \\ \wedge \ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge \ \mathsf{partitioned}(a, 0, j-1, j, j) \\ \wedge \ \mathsf{sorted}(a, i, |a|-1) \end{bmatrix}$$

$$\mu(\ell_f) = \mathsf{sorted}(\mathsf{rv}, 0, |a|-1)$$



BubbleSort: Verification Conditions

Six VCs:

$$\{\mu(\ell_0)\}\rho_{\pi_0}\{\mu(\ell_1)\}$$

$$\{\mu(\ell_1)\}\rho_{\pi_1^T}\{\mu(\ell_2)\}$$

$$\{\mu(\ell_1)\}\rho_{\pi_1^F}\{\mu(\ell_f)\}$$

$$\{\mu(\ell_2)\}\rho_{\pi_2^1}\{\mu(\ell_2)\}$$

$$\{\mu(\ell_2)\}\rho_{\pi_2^2}\{\mu(\ell_2)\}$$

$$\{\mu(\ell_2)\}\rho_{\pi_2^3}\{\mu(\ell_1)\}$$

where

$$\{\varphi\}\rho_\pi\{\psi\} \quad \Rightarrow \quad (\forall *)[\varphi \wedge \rho_\pi \ \rightarrow \ \psi']$$

Since all VCs are valid,
BubbleSort returns a sorted array.

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness
   (a) Annotations
⇒ (b) Function Calls
   (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures
   (a) Overview
   (b) Theories

BinarySearch: Function Calls

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$

ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$

`bool` BinarySearch(`int[]` a, `int` e) {

  ℓ_1: @ $0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$; $\quad \ldots \pi_0^a$

  `return` BSearch$(a, 0, |a| - 1, e)$; $\quad \ldots \pi_0$

}

Prove that if precondition is true, then

- precondition of BSearch holds in calling context;
- postcondition of BinarySearch holds if it terminates.

BinarySearch: Function Calls

m_0: @pre $0 \leq \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$
m_f: @post rv $\leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]$

```
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;                                   ... π1
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;                               ... π2
    else if (a[m] < e) {
      m1: @ 0 ≤ m + 1 ∧ u < |a| ∧ sorted(a, 0, |a| − 1);     ... π3^a
      return BSearch(a, m + 1, u, e);                        ... π3
    }
    else {
      m2: @ 0 ≤ ℓ ∧ m − 1 < |a| ∧ sorted(a, 0, |a| − 1);     ... π4^a
      return BSearch(a, ℓ, m − 1, e);                        ... π4
    }
  }
}
```

BinarySearch: Assertion Map

$$
\begin{aligned}
\mu(\ell_0) &: |a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(\ell_1) &: 0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(\ell_f) &: \mathsf{rv} \ \leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e] \\
\mu(m_0) &: 0 \leq \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_1) &: 0 \leq m + 1 \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_2) &: 0 \leq \ell \ \wedge \ m - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_f) &: \mathsf{rv} \ \leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]
\end{aligned}
$$

BinarySearch: Partial Correctness

Prove that if precondition is true, then

- postcondition of BSearch holds if it terminates:
  - if $\ell > u$ $\ldots \pi_1$
  - if $\ell \leq u \ \wedge \ a[\frac{\ell+u}{2}] = e$ $\ldots \pi_2$
  - if $\ell \leq u \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] < e$ $\ldots \pi_3$
  - if $\ell \leq u \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] \geq e$ $\ldots \pi_4$
- precondition of BSearch holds in calling contexts;
  - if $\ell \leq u \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] < e$ $\ldots \pi_3^a$
  - if $\ell \leq u \ \wedge \ a[\frac{\ell+u}{2}] \neq e \ \wedge \ a[\frac{\ell+u}{2}] \geq e$ $\ldots \pi_4^a$

BinarySearch: Function Call (Path π_3)

$$\rho_{\pi_3}: \quad \begin{array}{l} \ell \le u \ \wedge\ m' = \frac{\ell+u}{2} \ \wedge\ a[\frac{\ell+u}{2}] \neq e \ \wedge\ a[\frac{\ell+u}{2}] < e \\ \wedge\ \mathsf{rv}' = \textsc{BSearch}(a, \frac{\ell+u}{2}+1, u, e) \ \wedge\ \mathsf{pres}(\ldots) \end{array}$$

$$\Downarrow$$

$$\begin{array}{l} \ell \le u \ \wedge\ m' = \frac{\ell+u}{2} \ \wedge\ a[\frac{\ell+u}{2}] \neq e \ \wedge\ a[\frac{\ell+u}{2}] < e \\ \wedge\ \mathsf{rv}' = w \ \wedge\ w \leftrightarrow (\exists i)\left[\frac{\ell+u}{2}+1 \le i \le u \ \wedge\ a[i] = e\right] \ \wedge\ \mathsf{pres}(\ldots) \end{array}$$

Summarize function call with postcondition.

BinarySearch: Verification Conditions

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$

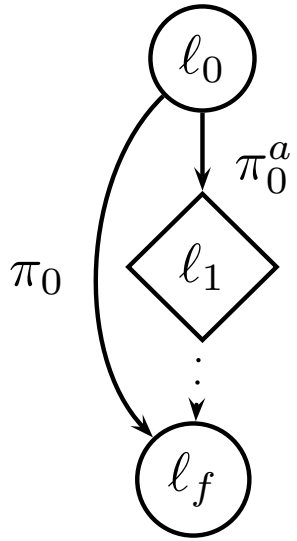
ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$

`bool` BinarySearch(`int[]` a, `int` e) {

  ℓ_1: @ $0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$; $\quad \ldots \pi_0^a$

  `return` BSearch$(a, 0, |a| - 1, e)$; $\quad \ldots \pi_0$

}



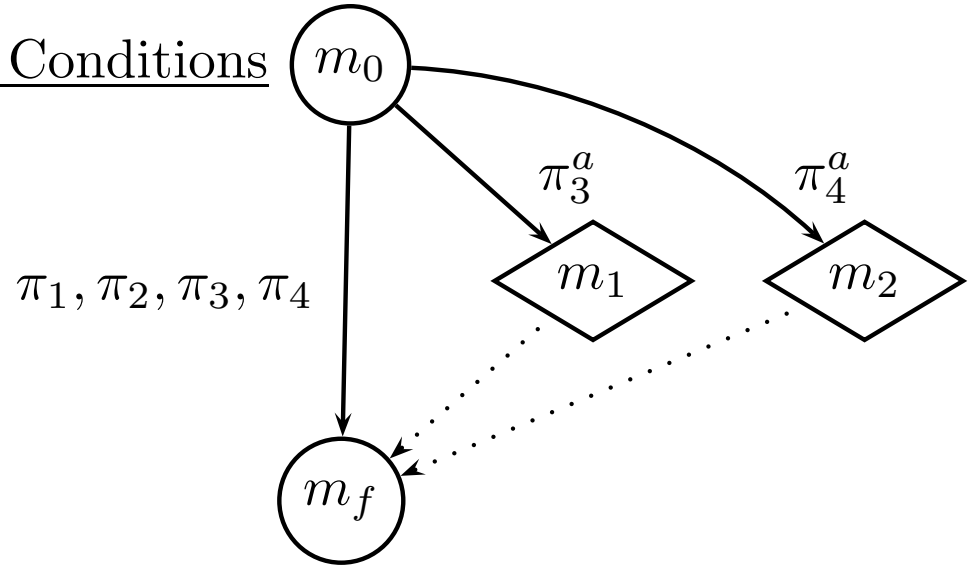
$$\{\mu(\ell_0)\}\rho_{\pi_0^a}\{\mu(\ell_1)\}:$$

$$(\forall *)\left[\begin{array}{l} |a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \ \rightarrow \\ \quad 0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \end{array}\right]$$

$$\{\mu(\ell_0)\}\rho_{\pi_0}\{\mu(\ell_f)\}:$$

$$(\forall *)\left[\begin{array}{l} |a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\ \wedge \ (w \leftrightarrow (\exists i)[0 \leq i \leq |a| - 1 \ \wedge \ a[i] = e]) \\ \quad \rightarrow \ (w \leftrightarrow (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]) \end{array}\right] \begin{array}{r} \mu(\ell_0) \\ \mu(m_f) \\ \mu(\ell_f) \end{array}$$

BinarySearch: Verification Conditions



$$\{\mu(m_0)\}\rho_{\pi_3^a}\{\mu(m_1)\}:$$

$$(\forall *)\begin{bmatrix} 0 \le \ell \ \wedge\ u < |a| \ \wedge\ \mathsf{sorted}(a,0,|a|-1) \\ \wedge\ \ell \le u \ \wedge\ m' = \frac{\ell+u}{2} \ \wedge\ a\left[\frac{\ell+u}{2}\right] \neq e \ \wedge\ a\left[\frac{\ell+u}{2}\right] < e \\ \quad \rightarrow\ 0 \le \frac{\ell+u}{2}+1 \ \wedge\ u < |a| \ \wedge\ \mathsf{sorted}(a,0,|a|-1) \end{bmatrix}$$

$$\{\mu(m_0)\}\rho_{\pi_4^a}\{\mu(m_2)\}:$$

$$(\forall *)\begin{bmatrix} 0 \le \ell \ \wedge\ u < |a| \ \wedge\ \mathsf{sorted}(a,0,|a|-1) \\ \wedge\ \ell \le u \ \wedge\ m' = \frac{\ell+u}{2} \ \wedge\ a\left[\frac{\ell+u}{2}\right] \neq e \ \wedge\ a\left[\frac{\ell+u}{2}\right] \ge e \\ \quad \rightarrow\ 0 \le \ell \ \wedge\ \frac{\ell+u}{2}-1 < |a| \ \wedge\ \mathsf{sorted}(a,0,|a|-1) \end{bmatrix}$$

BinarySearch: Verification Conditions

$\{\mu(\ell_0)\}\rho_{\pi_0^a}\{\mu(\ell_1)\}$

$\{\mu(\ell_0)\}\rho_{\pi_0}\{\mu(\ell_f)\}$

$\{\mu(m_0)\}\rho_{\pi_1}\{\mu(m_f)\}$

$\{\mu(m_0)\}\rho_{\pi_2}\{\mu(m_f)\}$

$\{\mu(m_0)\}\rho_{\pi_3^a}\{\mu(m_f)\}$

$\{\mu(m_0)\}\rho_{\pi_3}\{\mu(m_f)\}$

$\{\mu(m_0)\}\rho_{\pi_4^a}\{\mu(m_f)\}$

$\{\mu(m_0)\}\rho_{\pi_4}\{\mu(m_f)\}$

Since all VCs are valid,

BinarySearch returns **true** iff array a contains element e.

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
⇒ (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures
   (a) Overview
   (b) Theories

Intermediate Assertions: Runtime Assertions

- **Array Bounds**: If statement contains array access $a[i]$, add assertion

$$\texttt{@}\ (0 \leq i \leq |a| - 1)$$

- **Divide-by-Zero**: If statement contains
  - real division e_1/e_2
  - integer division e_1 `div` e_2
  - integer mod $e_1 \% e_2$

  add assertion

$$\texttt{@}\ (e_2 \neq 0)$$

Optional: to prove lack of runtime exceptions.

BubbleSort: Intermediate Assertions (Inner Loop)

$$\texttt{for } \ell_2\text{: } @ \begin{bmatrix} 1 \le i < |a| \ \wedge\ 0 \le j \le i \ \wedge\ |a| = |a|_0 \\ \wedge\ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \\ \wedge\ \mathsf{partitioned}(a, 0, j-1, j, j) \ \wedge\ \mathsf{sorted}(a, i, |a|-1) \end{bmatrix}$$

$(j := 0;\ j < i;\ j := j + 1)\ \{$

$\quad \ell_3^1\text{: } @\ 0 \le j < |a| \ \wedge\ 0 \le j+1 < |a|; \qquad \ldots \pi_3^1$

$\quad \texttt{if } (a[j] > a[j+1])\ \{$

$\quad\quad \ell_3^2\text{: } @\ 0 \le j < |a|; \qquad \ldots \pi_3^2$

$\quad\quad t := a[j];$

$\quad\quad \ell_3^3\text{: } @\ 0 \le j < |a| \ \wedge\ 0 \le j+1 < |a|; \qquad \ldots \pi_3^3$

$\quad\quad a[j] := a[j+1];$

$\quad\quad \ell_3^4\text{: } @\ 0 \le j+1 < |a|; \qquad \ldots \pi_3^4$

$\quad\quad a[j+1] := t;$

$\quad \}$

$\}$

Compiler optionally generates assertions.

BinarySearch: Intermediate Assertions

ℓ_0: @pre $|a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1)$

ℓ_f: @post rv $\leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e]$

`bool` BinarySearch(`int[]` a, `int` e) {

  ℓ_1: @ $0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1);$ $\quad \ldots \pi_0^a$

  `return` BSearch$(a, 0, |a| - 1, e);$ $\quad \ldots \pi_0$

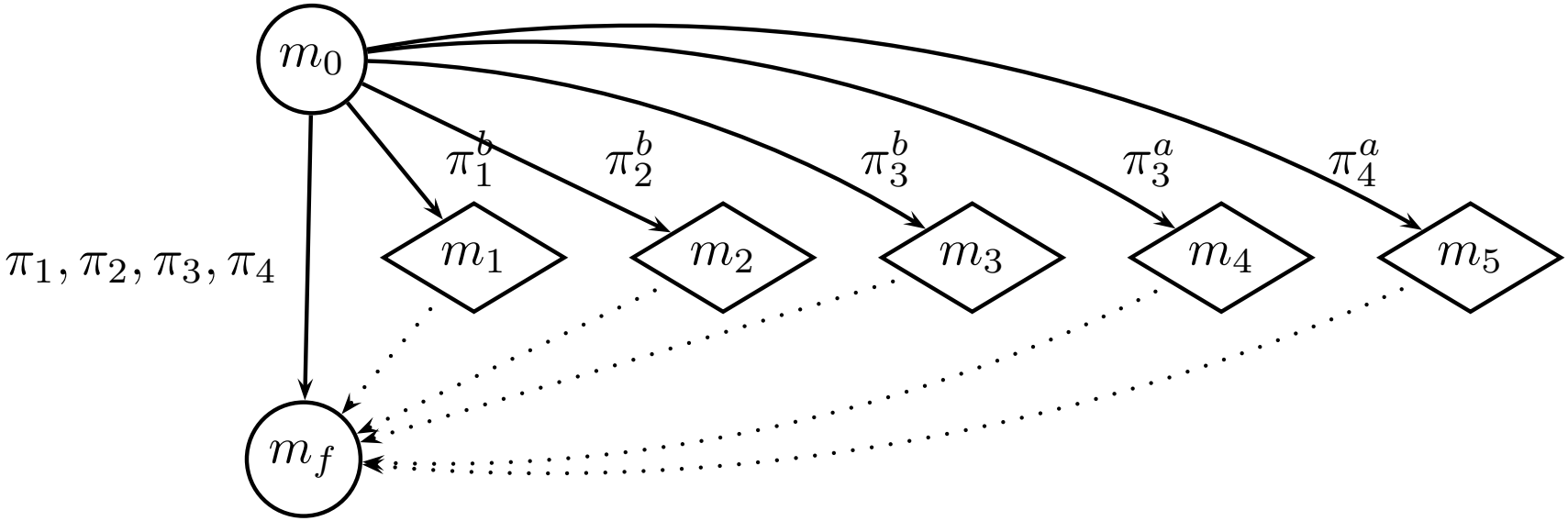
}

```
m_0: @pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, 0, |a| − 1)
m_f: @post rv ↔ (∃i)[ℓ ≤ i ≤ u ∧ a[i] = e]
bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;                                        ... π_1
  else {
    m_1: @ 2 ≠ 0;                                                 ... π_1^b
    m := (ℓ + u) div 2;
    m_2: @ 0 ≤ m < |a|;                                           ... π_2^b
    if (a[m] = e) return true;                                    ... π_2
    m_3: @ 0 ≤ m < |a|;                                           ... π_3^b
    else if (a[m] < e) {
      m_4: @ 0 ≤ m + 1 ∧ u < |a| ∧ sorted(a, 0, |a| − 1);         ... π_3^a
      return BSearch(a, m + 1, u, e);                             ... π_3
    }
    else {
      m_5: @ 0 ≤ ℓ ∧ m − 1 < |a| ∧ sorted(a, 0, |a| − 1);         ... π_4^a
      return BSearch(a, ℓ, m − 1, e);                             ... π_4
    }
  }
}
```

BinarySearch: Assertion Map

$$
\begin{aligned}
\mu(\ell_0) &: |a| \geq 0 \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(\ell_1) &: 0 \leq 0 \ \wedge \ |a| - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(\ell_f) &: \mathsf{rv} \ \leftrightarrow \ (\exists i)[0 \leq i < |a| \ \wedge \ a[i] = e] \\
\mu(m_0) &: 0 \leq \ell \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_1) &: 2 \neq 0 \\
\mu(m_2) &: 0 \leq m < |a| - 1 \\
\mu(m_3) &: 0 \leq m < |a| - 1 \\
\mu(m_4) &: 0 \leq m + 1 \ \wedge \ u < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_5) &: 0 \leq \ell \ \wedge \ m - 1 < |a| \ \wedge \ \mathsf{sorted}(a, 0, |a| - 1) \\
\mu(m_f) &: \mathsf{rv} \ \leftrightarrow \ (\exists i)[\ell \leq i \leq u \ \wedge \ a[i] = e]
\end{aligned}
$$

BinarySearch: Augmented Transition System



$$\rho_{\pi_1^b}: \quad \ell \leq u \ \wedge \ \mathsf{pres}(\ldots)$$

$$\rho_{\pi_2^b}: \quad \ell \leq u \ \wedge \ m' = \tfrac{\ell+u}{2} \ \wedge \ \mathsf{pres}(\ldots)$$

$$\rho_{\pi_3^b}: \quad \ell \leq u \ \wedge \ m' = \tfrac{\ell+u}{2} \ \wedge \ a\left[\tfrac{\ell+u}{2}\right] \neq e \ \wedge \ \mathsf{pres}(\ldots)$$

BinarySearch: Verification Conditions

$$\{\mu(\ell_0)\}\rho_{\pi_0^a}\{\mu(\ell_1)\} \qquad \{\mu(m_0)\}\rho_{\pi_1}\{\mu(m_f)\}$$

$$\{\mu(\ell_0)\}\rho_{\pi_0}\{\mu(\ell_f)\} \qquad \{\mu(m_0)\}\rho_{\pi_1^b}\{\mu(m_1)\}$$

$$\{\mu(m_0)\}\rho_{\pi_2^b}\{\mu(m_2)\}$$

$$\{\mu(m_0)\}\rho_{\pi_2}\{\mu(m_f)\}$$

$$\{\mu(m_0)\}\rho_{\pi_3^b}\{\mu(m_3)\}$$

$$\{\mu(m_0)\}\rho_{\pi_3^a}\{\mu(m_4)\}$$

$$\{\mu(m_0)\}\rho_{\pi_3}\{\mu(m_f)\}$$

$$\{\mu(m_0)\}\rho_{\pi_4^a}\{\mu(m_5)\}$$

$$\{\mu(m_0)\}\rho_{\pi_4}\{\mu(m_f)\}$$

Since all VCs are valid,

BinarySearch is also correct w.r.t. runtime assertions.

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
   (c) Runtime Assertions

⇒ 3. Total Correctness (Termination)

4. Decision Procedures
   (a) Overview
   (b) Theories

BubbleSort: Motivation

```
int[] BubbleSort(int[] a) {
  int i, j, t;
  for (i := |a| − 1; i > 0; i := i − 1) {
    for (j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Does BubbleSort always terminate?

BinarySearch: Motivation

```
bool BinarySearch(int[] a, int e) {
  return BSearch(a, 0, |a| − 1, e);
}

bool BSearch(int[] a, int ℓ, int u, int e) {
  int m;
  if (ℓ > u) return false;
  else {
    m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BSearch(a, m + 1, u, e);
    else return BSearch(a, ℓ, m − 1, e);
  }
}
```

Does BinarySearch always terminate?

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
   (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures

⇒ (a) Overview
   (b) Theories

Motivation

Is this verification condition valid?

$$
(\forall *)\ \left[\begin{array}{l}
1 \le i < |a| \ \wedge\ 0 \le j \le i \ \wedge\ |a| = |a|_0 \\
\wedge\ \mathsf{partitioned}(a, 0, i, i+1, |a|-1) \\
\wedge\ \mathsf{partitioned}(a, 0, j-1, j, j) \\
\wedge\ \mathsf{sorted}(a, i, |a|-1) \\
\wedge\ j < i \ \wedge\ a[j] > a[j+1] \\
\quad \rightarrow \begin{array}{l}
1 \le i < |a| \ \wedge\ 0 \le j+1 \le i \ \wedge\ |a| = |a|_0 \\
\wedge\ \mathsf{partitioned}(a[j : a[j+1]][j+1 : a[j]], 0, i, i+1, |a|-1) \\
\wedge\ \mathsf{partitioned}(a[j : a[j+1]][j+1 : a[j]], 0, j, j+1, j+1) \\
\wedge\ \mathsf{sorted}(a[j : a[j+1]][j+1 : a[j]], i, |a|-1)
\end{array}
\end{array}\right]
$$

How do we prove it?

Use **decision procedures**!

Motivation

Goal: Automate proofs of correctness.

Method:

- The VC-method reduces program correctness to proving verification conditions, *i.e.*, first-order validities.
- Now focus on automatic reasoning.

Propositional Logic (PL)

- constants **true**, **false**
- propositional variables $P, Q, R, \ldots$
- connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

Sentence (Formula):
A constant, a variable, or a connection of two propositional sentences.

Examples:

$$(P \ \wedge \ Q) \ \rightarrow \ R$$

$$\neg(P \wedge Q) \ \leftrightarrow \ (\neg P \ \vee \ \neg Q)$$

$$(\neg P \ \vee \ Q) \ \leftrightarrow \ (P \ \rightarrow \ Q)$$

Propositional Logic (PL)

Interpretation I:
Assignment of each propositional variables to one of $\{\mathbf{true}, \mathbf{false}\}$.

Propositional sentence φ is

- **valid** if φ evaluates to **true** under all interpretations;
- **satisfiable** if φ evaluates to **true** under some interpretation;
- **unsatisfiable** if φ evaluates to **false** under all interpretations.

So φ is valid iff $\neg\varphi$ is unsatisfiable.

First-Order Logic (FOL)

- constants **true**, **false**
- n-ary predicates $p, q, r, \ldots$
- n-ary functions $f, g, \ldots$
- variables, $x, y, z, \ldots$
- connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- quantifiers $\forall$, $\exists$

Term: A variable or a function application. x, $f(x)$, $f(x, g(x, y))$

Atom: A constant or predicate. **true**, $p(f(x), y)$

Literal: An atom or its negations. $p(x, g(y))$, $\neg p(x, g(y))$

First-Order Logic (FOL)

Sentence (Formula):

A literal, a connection of two sentences,

or a quantification of a sentence.

A sentence is **closed** if all variables are quantified.

Examples:

$$(\forall x)[p(x, g(y)) \ \wedge \ q(y)]$$

$$(\exists y)(\forall x)[p(x, g(y)) \ \wedge \ q(y)]$$

$$(\forall x)(\exists y)[p(x, y) \ \wedge \ p(x, f(f(y)))]$$

First-Order Logic (FOL)

Interpretation $I : \langle D, \alpha \rangle$:
Assignment by α of all

- variables to domain D
- n-ary predicates to $D^n \to \{\mathbf{true}, \mathbf{false}\}$
- n-ary functions to $D^n \to D$

FOL sentence φ is

- **valid** if φ evaluates to **true** under all interpretations;
- **satisfiable** if φ evaluates to **true** under some interpretation;
- **unsatisfiable** if φ evaluates to **false** under all interpretations.

So φ is valid iff $\neg\varphi$ is unsatisfiable.

Theory

A **theory** T is a set of FOL closed sentences ("axioms").
FOL sentence φ is

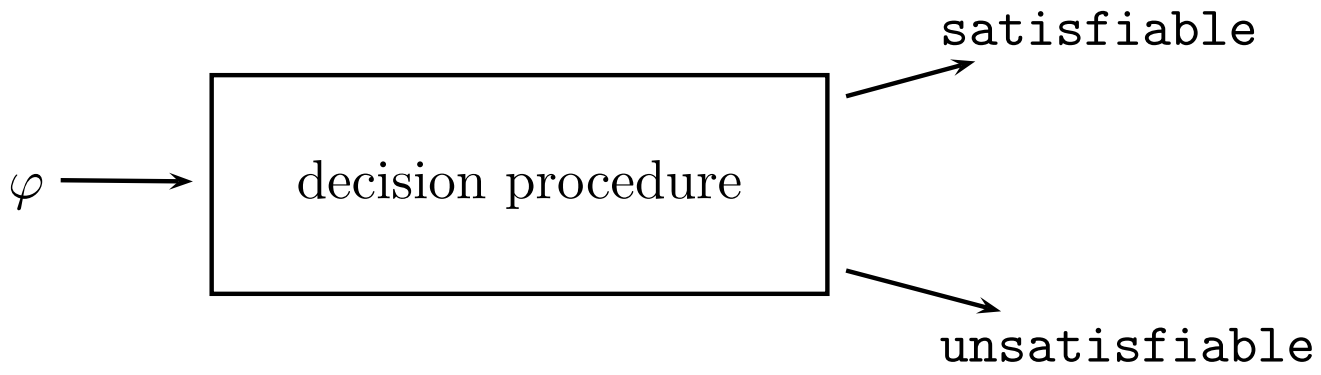
- T-**valid** if φ evaluates to **true** under
  all interpretations satisfying T
  (*i.e.*, interpretations for which each axiom of T is satisfied);
- T-**satisfiable** if φ evaluates to **true** under
  some interpretation satisfying T;
- T-**unsatisfiable** if φ evaluates to **false** under
  all interpretations satisfying T.

So φ is T-valid iff $\neg\varphi$ is T-unsatisfiable.

What is a Decision Procedure?

Theory T: collection of formulas (**axioms**)

Decision procedure for T: algorithm for deciding whether or not a T-formula φ is satisfiable in T

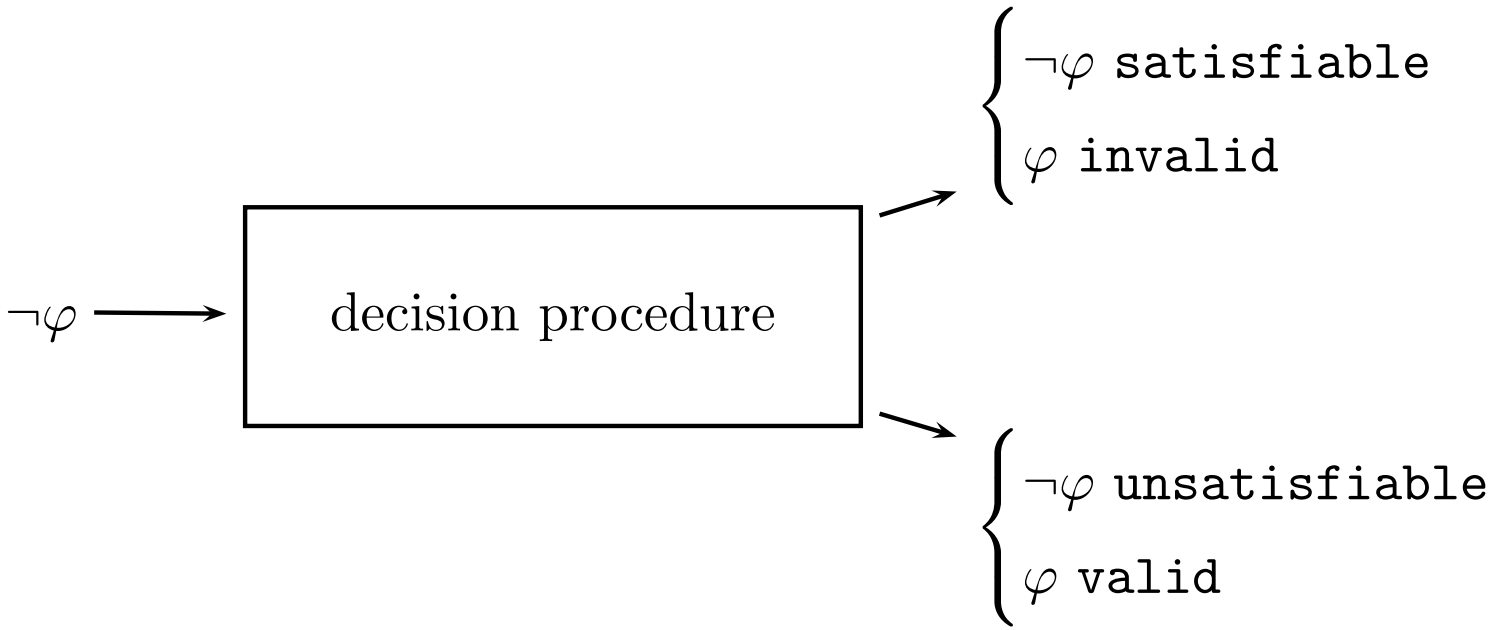


- always terminates
- with the right answer

T is said to be a **decidable theory**

What is a Decision Procedure?

Same algorithm can be used for deciding whether or not
a T-formula φ is valid in T:



Advantages of Decision Procedures

Efficiency

Using decision procedures is more effective than encoding the axioms of the theory and employing first-order logic reasoning.

Availability

Decision procedures are available for many useful theories.

Applications

Decision procedures have been used in

- theorem proving
- model checking
- verification
- synthesis

$\vdots$

Decision Procedure

Algorithm for deciding satisfiability of sentences in some theory.

- Propositional Logic (PL) is **decidable**.
  - Algorithms exist for deciding satisfiability of PL sentences, *e.g.*, truth table, Davis-Putnam-Logemann-Loveland (DPLL), ...
- First-order Logic (FOL) is **not decidable**.
  - No algorithm exists to decide satisfiability of first-order sentences.
  - There are decidable **fragments** of FOL:
    - specific theories
    - restrictions on use of quantifiers

Examples: Decidable Theories

- Integer Linear (Presburger) Arithmetic, $T_{\mathbb{Z}}$

$$(\forall y)(\exists x)(y = 2x) \qquad T_{\mathbb{Z}}\text{-invalid}$$

- Real Linear Arithmetic, $T_{\mathbb{R}}$

$$x > 0 \ \wedge \ y > 0 \ \wedge \ x + y = 1 \qquad T_{\mathbb{R}}\text{-sat}$$

- Equality, $T_{\mathbb{E}}$

$$f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge f(a) \neq a \qquad T_{\mathbb{E}}\text{-unsat}$$

- Recursive Data Structures, $T_{\mathbb{D}}$

$$\mathsf{car}(\mathsf{cons}(x, y)) \neq x \qquad T_{\mathbb{D}}\text{-unsat}$$

Examples: Decidable Theories

- Arrays, $T_{\mathbb{A}}$

$$\begin{array}{r} \mathsf{sorted}(0, 5, a[0:7][5:9]) \\ \wedge\ \mathsf{sorted}(0, 5, a[0:11][5:13]) \end{array} \qquad T_{\mathbb{A}}\text{-unsat}$$

- Combinations of theories

$$1 \leq x \ \wedge\ x \leq 2 \ \wedge\ f(x) \neq f(1) \ \wedge\ f(x) \neq f(2) \quad T_{\mathbb{Z}} \cup T_{\mathbb{E}}\text{-unsat}$$

Outline

0. Motivation
1. Transition Systems
2. Partial Correctness
   (a) Annotations
   (b) Function Calls
   (c) Runtime Assertions
3. Total Correctness (Termination)
4. Decision Procedures
   (a) Overview
⇒ (b) Theories

Theory of Equality $T_{\mathbb{E}}$

(with variables, quantifiers, and logical connectives)

$\Sigma = \{a, b, c, \ldots, f, g, h, \ldots, p, q, r, \ldots, =\}$ "signature"

Uninterpreted symbols:

- Constants: $a, b, c, \ldots$
- Functions: $f, g, h, \ldots$
- Predicates: $p, q, r, \ldots$

Examples:

$f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge f(a) \neq a$ $T_{\mathbb{E}}$-unsatisfiable

$x = y \ \wedge \ f(x) \neq f(y)$ $T_{\mathbb{E}}$-unsatisfiable

$(\forall x)(\exists y)[x = f(y)]$ $T_{\mathbb{E}}$-satisfiable

Theory of Equality $T_{\mathbb{E}}$

Axiom schema:

$$(\forall x)[x = x] \qquad \textbf{(reflexivity)}$$

$$(\forall x, y)[x = y \rightarrow y = x] \qquad \textbf{(symmetry)}$$

$$(\forall x, y, z)[x = y \wedge y = z \rightarrow x = z] \qquad \textbf{(transitivity)}$$

$$(\forall \overrightarrow{x}, \overrightarrow{y}) \left[\bigwedge_i x_i = y_i \rightarrow f(\overrightarrow{x}) = f(\overrightarrow{y}) \right] \qquad \textbf{(congruence)}$$

Theory of Equality $T_{\mathbb{E}}$

- Full: undecidable [Church, 36] [Turing, 36]
- Quantifier-free: decidable [Ackerman, 54]
  - Efficient algorithms based on Congruence Closure (++)
    [Shostak, 78]
    [Downey, Sethi and Tarjan, 80]
    [Nelson and Oppen, 80]

Theory of Integers $T_{\mathbb{Z}}$ (Presburger Arithmetic)

$\Sigma = \{0, 1, +, -, =, <\}$

- Domain: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- Constants: n for each $n \in \mathbb{Z}$
- Functions: $+$ (addition), $-$ (subtraction)
- Predicates: $=$ (equality), $<$ (comparison)

Examples:

$$(\forall x)(\exists y)[y = 2x] \qquad T_{\mathbb{Z}}\text{-valid}$$

$$(\forall x)(\exists y)[x = 2y] \qquad T_{\mathbb{Z}}\text{-invalid}$$

$$x > 0 \ \wedge \ y > 0 \ \wedge \ x + y = 1 \qquad T_{\mathbb{Z}}\text{-unsatisfiable}$$

Theory of Integers $T_{\mathbb{Z}}$ (Presburger Arithmetic)

- Full: decidable
  - Quantifier-elimination [Presburger, 29] [Cooper, 72] [Fisher & Rabin, 74] (– –)
- Quantifier-free: decidable [Papadimitriou, 81] (–)
  - Omega test [Pugh, 94] (+)
- Introduce $\times$ (multiplications):

$$(\exists x, y, z)[x^3 + y^3 = z^3] \qquad T_{\mathbb{Z}}\text{-unsatisfiable}$$

  - undecidable [Gödel, 31] [Church, 36]
  - even for single quantifier-free equation [Matiyasevich, 70]

Theory of Reals $T_{\mathbb{R}}$

$\Sigma = \{0, 1, +, -, =, <\}$

- Domain: $\mathbb{R} = \{\ldots, 0, \ldots, \frac{3}{2}, \ldots\}$
- Constants: n for each $n \in \mathbb{Z}$
- Functions: $+$ (addition), $-$ (subtraction)
- Predicates: $=$ (equality), $<$ (comparison)

Examples:

$$(\forall x)(\exists y)[x = 2y] \qquad T_{\mathbb{R}}\text{-valid}$$

$$x > 0 \ \wedge \ y > 0 \ \wedge \ x + y = 1 \qquad T_{\mathbb{R}}\text{-satisfiable}$$

Theory of Reals $T_{\mathbb{R}}$

- Full: decidable
  - Quantifier-elimination [Tarski, 51] (– –)
  - Cylindrical algebraic decomposition [Collins, 75] (–)
- Quantifier-free: decidable (many methods)
  - Fourier-Motzkin [Lassez & Mahler, 92] (+)
  - Simplex [Dantzig, 61] (++)
  - [Kachiyan, 79] (++), [Karmarker, 84] (++)

Theory of Reals $T_{\mathbb{R}}$ (with Multiplication)

Introduce $\times$ (multiplication):
Full: decidable [Tarski, 51] (– –)

- Cylindrical algebraic decomposition [Collins, 75] (–)
- Inherently doubly-exponential [Davenport & Heintz, 88]

Remark:
If we add ceiling

$$\lceil x \rceil = \min\{y \ : \ y \in \mathbb{Z} \ \wedge \ y \geq x\}$$

or floor

$$\lfloor x \rfloor = \max\{y \ : \ y \in \mathbb{Z} \ \wedge \ y \leq x\}$$

then we are able to encode integers, and we lose decidability.

Theory of Reals $T_{\mathbb{R}}$

Why are **constants** $n \in \mathbb{Z}$?
For atom $\alpha < \beta$ (similarly, $\alpha = \beta$),

- Collect denominators d_i of fractional coefficients.
- Multiply every coefficient by their product:

$$\alpha \left(\prod_i d_i \right) < \beta \left(\prod_i d_i \right)$$

- Coefficients are now integers.

Example:

$$\frac{3}{2}x < \frac{4}{3}y \quad \Rightarrow \quad 9x < 8y$$

Theory of Reals $T_{\mathbb{R}}$

Why theory of **reals**?

- Linear case: cannot express irrational numbers.
  If there is an irrational solution,
  then there is a rational solution.
- Polynomial case:

$$x \cdot x = 2$$

  expresses $x = \sqrt{2}$, which is irrational.

Theory of Recursive Data Structures (RDS) $T_{\mathbb{D}}$

Parametric theory. Each RDS has

- n_C-ary constructor $C(\overrightarrow{x})$
- n_C projection functions π_i^C
- one atom predicate $atom_C$

Axiom schema: axioms of $T_{\mathbb{E}}$ +

$(\forall \overrightarrow{x})[\pi_i(C(\overrightarrow{x})) = x_i]$ (**projection**)

$(\forall x)[\neg atom_C(x) \rightarrow C(\ldots, \pi_i(x), \ldots) = x]$ (**construction**)

$(\forall \overrightarrow{x})[\neg atom_C(C(\overrightarrow{x}))]$ (**atom**)

Example: List RDS

Define

- constructor cons
- projection functions car, cdr
- atom predicate atom

Axiom schema: axioms of $T_{\mathbb{E}}$ +

$$(\forall x, y)[\mathsf{car}(\mathsf{cons}(x, y)) = x]$$
$$(\forall x, y)[\mathsf{cdr}(\mathsf{cons}(x, y)) = y]$$
$$(\forall x)[\neg\mathsf{atom}(x) \rightarrow \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) = x]$$
$$(\forall x, y)[\neg\mathsf{atom}(\mathsf{cons}(x, y))]$$

Theory of Recursive Data Structures (RDS) $T_{\mathbb{D}}$

- Full: decidable (– –)
  - Quantifier-elimination over term algebras [Mal'cev, 71] [Hodges, 93]
  - Via *pairing functions* [Tenney, 72] [Oppen, 80]
  - Not elementary recursive [Tenny, 72]
- Quantifier-free: decidable [Oppen, 80] (++)
  - if values of projection functions on atoms are defined: NP-complete [Oppen, 80]

Decidable Domains

There is a decision procedure for deciding the validity of quantified sentences over this domain.

Examples:

- abelian groups
- term algebras
- boolean algebras
- Presburger arithmetic
- real numbers
- dense linear order

  ⋮

Abelian Groups

$\mathcal{G}_1:$	$(\forall x)[x \circ e = x]$ $(\textit{right identity})$
$\mathcal{G}_2:$	$(\forall x)[x \circ x^{-1} = e]$ $(\textit{right inverse})$
$\mathcal{G}_3:$	$(\forall x, y, z)\,[(x \circ y) \circ z = x \circ (y \circ z)]$ $(\textit{associativity})$
$\mathcal{G}_4:$	$(\forall x, y)\,[x \circ y = y \circ x]$ $(\textit{commutativity})$

The theory $\mathcal{G}_1 + \mathcal{G}_2 + \mathcal{G}_3 + \mathcal{G}_4$ is decidable.

Remark:

The theory $\mathcal{G}_1 + \mathcal{G}_2 + \mathcal{G}_3$ is **not** decidable.

Abelian Groups

$$(\forall x, y, z)\left[\begin{array}{l} x \circ y = e \\ \rightarrow\ \ y \circ (z \circ x) = z \end{array}\right]$$

$\downarrow$

Decision Procedure $\rightarrow$ "`valid`"

$$(\forall x, y)\left[\begin{array}{l} x \circ y = e \\ \rightarrow\ \ (x = e\ \ \vee\ \ y = e) \end{array}\right]$$

$\downarrow$

Decision Procedure $\rightarrow$ "`not valid`"

Dense Linear Order without Endpoints

$x \preceq x$ (*reflexivity*)

$x \preceq y \ \wedge \ y \preceq z \ \rightarrow \ x \preceq z$ (*transitivity*)

$x \preceq y \ \wedge \ y \preceq x \ \rightarrow \ x = y$ (*antisymmetry*)

$x \preceq y \ \wedge \ y \preceq x$ (*total order*)

$(\forall x)(\exists y)(\exists z)[y \preceq x \ \wedge \ x \preceq z]$ (*without endpoints*)

$x \preceq y \ \rightarrow \ (\exists z)[x \preceq z \ \wedge z \ \preceq y]$ (*denseness*)

The theory of dense linear order without endpoints is decidable.

Dense Linear Order without Endpoints

$$(\forall x, y)\left[\begin{array}{l} x \preceq y \\ \rightarrow\ (\exists z_1, z_2)[x \preceq z_1 \preceq z_2 \preceq y] \end{array}\right]$$

$\downarrow$

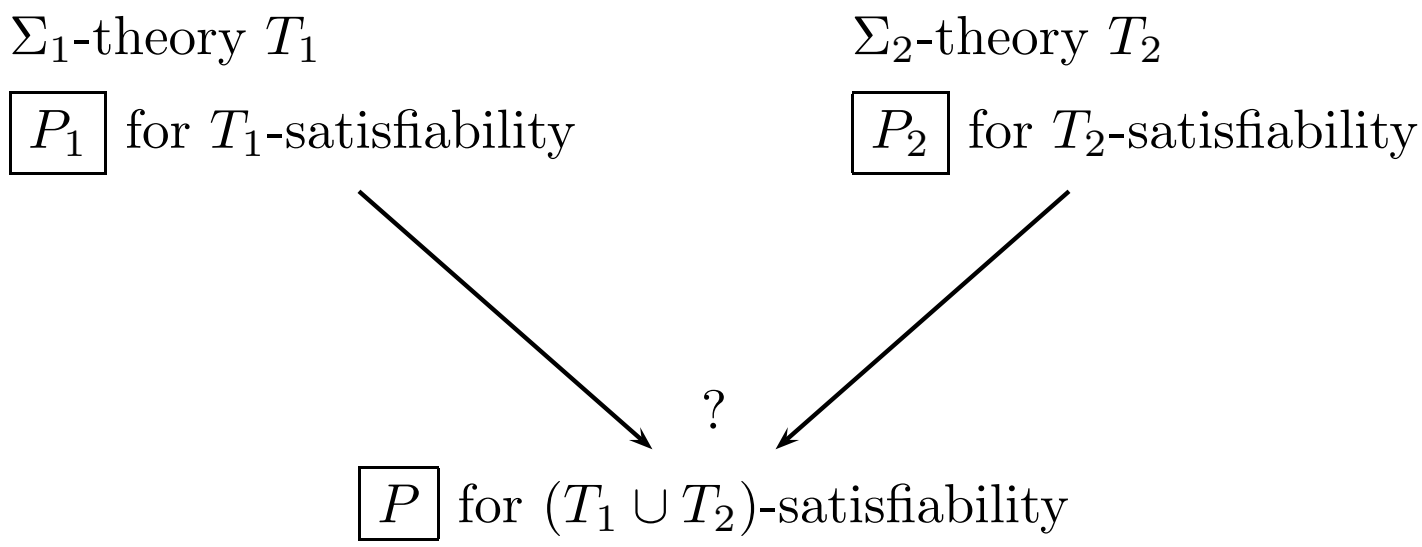
Decision Procedure $\rightarrow$ "`valid`"

$$(\forall x, y)\left[\ (\exists x)(\forall y)[y \preceq x]\ \right]$$

$\downarrow$

Decision Procedure $\rightarrow$ "`not valid`"

Combining Decision Procedures



Problem:

Decision procedures are domain specific.

How do we combine them?

Example:

$1 \leq x \ \wedge \ x \leq 2 \ \wedge \ f(x) \neq f(1) \ \wedge \ f(x) \neq f(2)$ $\qquad$ $(T_{\mathbb{E}} \cup T_{\mathbb{Z}})$-unsat

Nelson-Oppen Combination Method

$$\Sigma_1 \cap \Sigma_2 = \emptyset$$

Σ_1-theory T_1

stably infinite

$\boxed{P_1}$ for T_1-satisfiability

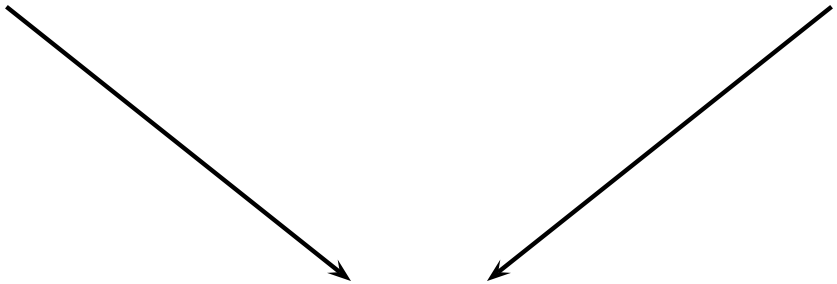
of quantifier-free Σ_1-formulae

Σ_2-theory T_2

stably infinite

$\boxed{P_2}$ for T_2-satisfiability

of quantifier-free Σ_2-formulae



$\boxed{P}$ for $(T_1 \cup T_2)$-satisfiability

of quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formulae

Theory of Arrays $T_{\mathbb{A}}$

$\Sigma = \Sigma_{\mathbb{Z}} \cup \Sigma_{\mathsf{elem}} \cup \{\cdot[\cdot],\ \cdot[\cdot:\cdot]\}$

- Parameter theory: Element theory T_{elem}
- Domain: $\boxed{\mathbb{A} = \{\mathbb{Z} \to \mathsf{elem}\}}$
- Constants: constants of $T_{\mathbb{Z}}$ and T_{elem}
- Functions: functions of $T_{\mathbb{Z}}$ and T_{elem}
  and $\cdot[\cdot]$ (read), $\cdot[\cdot:\cdot]$ (write)
- Predicates: predicates of $T_{\mathbb{Z}}$ and T_{elem}

Example:

$$\begin{array}{r} \mathsf{sorted}(0, 5, a[0:7][5:9]) \\ \wedge\ \mathsf{sorted}(0, 5, a[0:11][5:13]) \end{array} \qquad T_{\mathbb{A}}\text{-unsatisfiable}$$

where $\mathsf{sorted}(\ell, u, a) \stackrel{\text{def}}{=} (\forall i, j)[\ell \le i \le j \le u \ \rightarrow\ a[i] \le a[j]]$

Theory of Arrays $T_{\mathbb{A}}$

Axiom:

$$(\forall \text{ arrays } a)(\forall \text{ elem } e)(\forall\, i, j \in \mathbb{Z})$$

$$\left[\begin{array}{rcl} & i = j \;\rightarrow\; a[i:e][j] = e \\ \wedge & i \neq j \;\rightarrow\; a[i:e][j] = a[j] \end{array}\right] \qquad \text{(read-over-write)}$$

Theory of Arrays $T_{\mathbb{A}}$

Satisfiability:

- Full: undecidable
- Quantifier-free: decidable
  - read-over-write ([McCarthy, 62])
    with combination of theories ([Nelson & Oppen, 79])
- Quantifier-free with = (extensional theory): decidable
  - [Stump, Barrett, Dill & Levitt, 01]
- **Array property fragment**: decidable
  - One alternation of quantifiers, with syntactic constraints
  - Assumption:
    Quantifier-free combination of $T_{\mathbb{E}} \cup T_{\mathbb{Z}} \cup T_{\mathsf{elem}}$ is decidable
  - [Bradley, Manna & Sipma 2005]