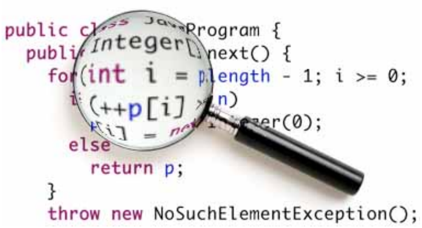
15-819M: Data, Code, Decisions

11: Proving Loop Properties

André Platzer

aplatzer@cs.cmu.edu
Carnegie Mellon University, Pittsburgh, PA



Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations?

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$
- 10 iterations?

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$
- 10 iterations? Unwind $11\times$

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$
- 10 iterations? Unwind $11\times$
- 10000 iterations?

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \{\texttt{p};\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$
- 10 iterations? Unwind $11\times$
- 10000 iterations? Unwind $10001\times$
  (and don't make any plans for the rest of the day)

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{if}\ \texttt{(b)}\ \texttt{\{p;}\ \textbf{while}\ \texttt{(b)}\ \texttt{p\}}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind 1×
- 10 iterations? Unwind 11×
- 10000 iterations? Unwind 10001×
  (and don't make any plans for the rest of the day)
- an unknown number of iterations?

Proving Loop

Symbolic execution of loops: unwind

$$\text{unwindLoop}\ \frac{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{if}\ \ \texttt{(b)}\ \ \texttt{\{p;}\ \ \mathbf{while}\ \ \texttt{(b)}\ \texttt{p\}}\ \omega]\phi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta}$$

How to handle a loop with...

- 0 iterations? Unwind $1\times$
- 10 iterations? Unwind $11\times$
- 10000 iterations? Unwind $10001\times$
  (and don't make any plans for the rest of the day)
- an unknown number of iterations?

We need an invariant rule (or some other form of induction)

Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Loop Invariants

Idea behind loop invariants

- A formula *Inv* whose validity is preserved by loop guard and body
- Consequence: if *Inv* was valid at start of the loop, then it still holds after arbitrarily many loop iterations
- If the loop terminates at all, then *Inv* holds afterwards
- Encode the desired postcondition after loop into *Inv*

Loop Invariants

Idea behind loop invariants

- A formula *Inv* whose validity is preserved by loop guard and body
- Consequence: if *Inv* was valid at start of the loop, then it still holds after arbitrarily many loop iterations
- If the loop terminates at all, then *Inv* holds afterwards
- Encode the desired postcondition after loop into *Inv*

Basic Invariant Rule

loopInvariant $\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta$

Loop Invariants

Idea behind loop invariants

- A formula *Inv* whose validity is preserved by loop guard and body
- Consequence: if *Inv* was valid at start of the loop, then it still holds after arbitrarily many loop iterations
- If the loop terminates at all, then *Inv* holds afterwards
- Encode the desired postcondition after loop into *Inv*

Basic Invariant Rule

$$\Gamma \Longrightarrow \mathcal{U}\,\mathit{Inv}, \Delta \qquad \text{(initially valid)}$$

loopInvariant

$$\Gamma \Longrightarrow \mathcal{U}[\pi\,\mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta$$

Loop Invariants

Idea behind loop invariants

- A formula *Inv* whose validity is preserved by loop guard and body
- Consequence: if *Inv* was valid at start of the loop, then it still holds after arbitrarily many loop iterations
- If the loop terminates at all, then *Inv* holds afterwards
- Encode the desired postcondition after loop into *Inv*

Basic Invariant Rule

$$\text{loopInvariant}\quad \frac{\begin{array}{ll} \Gamma \Longrightarrow \mathcal{U}\,\mathit{Inv}, \Delta & \text{(initially valid)} \\ \mathit{Inv},\ b \doteq \mathtt{TRUE} \Longrightarrow [\mathrm{p}]\mathit{Inv} & \text{(preserved)} \end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \ (\mathtt{b})\ \mathrm{p}\ \omega]\phi, \Delta}$$

Loop Invariants

Idea behind loop invariants

- A formula *Inv* whose validity is preserved by loop guard and body
- Consequence: if *Inv* was valid at start of the loop, then it still holds after arbitrarily many loop iterations
- If the loop terminates at all, then *Inv* holds afterwards
- Encode the desired postcondition after loop into *Inv*

Basic Invariant Rule

$$\text{loopInvariant}\ \frac{\begin{array}{ll}\Gamma \Longrightarrow \mathcal{U}\,\mathit{Inv}, \Delta & \text{(initially valid)}\\ \mathit{Inv},\ b \doteq \mathtt{TRUE} \Longrightarrow [\mathrm{p}]\mathit{Inv} & \text{(preserved)}\\ \mathit{Inv},\ b \doteq \mathtt{FALSE} \Longrightarrow [\pi\ \omega]\phi & \text{(use case)}\end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \ \mathtt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

Loop Invariants

Basic Invariant Rule: Problem

$$\text{loopInvariant}\ \frac{\begin{array}{ll}\Gamma \Longrightarrow \mathcal{U}\, \mathit{Inv}, \Delta & \text{(initially valid)} \\ \mathit{Inv},\ b \doteq \texttt{TRUE} \Longrightarrow [\mathrm{p}]\mathit{Inv} & \text{(preserved)} \\ \mathit{Inv},\ b \doteq \texttt{FALSE} \Longrightarrow [\pi\ \omega]\phi & \text{(use case)}\end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

Loop Invariants

Basic Invariant Rule: Problem

$$\text{loopInvariant}\ \frac{\begin{array}{ll}\Gamma \Longrightarrow \mathcal{U}\mathit{Inv}, \Delta & \text{(initially valid)}\\ \mathit{Inv},\ b \doteq \mathtt{TRUE} \Longrightarrow [\mathrm{p}]\mathit{Inv} & \text{(preserved)}\\ \mathit{Inv},\ b \doteq \mathtt{FALSE} \Longrightarrow [\pi\ \omega]\phi & \text{(use case)}\end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \mathtt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

- Context Γ, Δ, $\mathcal{U}$ must be omitted in 2nd and 3rd premise:

  Γ, Δ in general don't hold in state defined by $\mathcal{U}$

  2nd premise *Inv* must be invariant for any state, not only $\mathcal{U}$

  3rd premise We don't know the state after the loop exits

Loop Invariants

Basic Invariant Rule: Problem

$$\text{loopInvariant}\ \frac{\begin{array}{ll}\Gamma \Longrightarrow \mathcal{U}\,Inv, \Delta & \text{(initially valid)}\\ Inv,\ b \doteq \mathtt{TRUE} \Longrightarrow [\mathrm{p}]Inv & \text{(preserved)}\\ Inv,\ b \doteq \mathtt{FALSE} \Longrightarrow [\pi\ \omega]\phi & \text{(use case)}\end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \mathtt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

- Context Γ, Δ, $\mathcal{U}$ must be omitted in 2nd and 3rd premise:

  Γ, Δ in general don't hold in state defined by $\mathcal{U}$

  2nd premise *Inv* must be invariant for any state, not only $\mathcal{U}$

  3rd premise We don't know the state after the loop exits

- But: context contains (part of) precondition and class invariants

Loop Invariants

Basic Invariant Rule: Problem

$$\text{loopInvariant}\ \frac{\begin{array}{ll}\Gamma \Longrightarrow \mathcal{U}\, \mathit{Inv}, \Delta & \text{(initially valid)}\\ \mathit{Inv},\ b \doteq \texttt{TRUE} \Longrightarrow [\mathrm{p}]\mathit{Inv} & \text{(preserved)}\\ \mathit{Inv},\ b \doteq \texttt{FALSE} \Longrightarrow [\pi\ \omega]\phi & \text{(use case)}\end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

- Context Γ, Δ, $\mathcal{U}$ must be omitted in 2nd and 3rd premise:

  Γ, Δ in general don't hold in state defined by $\mathcal{U}$

  2nd premise *Inv* must be invariant for any state, not only $\mathcal{U}$

  3rd premise We don't know the state after the loop exits
- But: context contains (part of) precondition and class invariants
- Required context information must be added to loop invariant *Inv*

Example

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \mathbin{-\!\!>} \mathtt{a}[x] \doteq 1)$

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\, \textbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \rightarrow \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \leq \mathtt{i}\ \ \&\ \ \mathtt{i} \leq \mathtt{a.length}$

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\mathbf{int}\ x;\ (0 \le x < \mathtt{a.length} \ \text{->}\ \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \le \mathtt{i}\ \ \&\ \ \mathtt{i} \le \mathtt{a.length}$

$\&\ \ \forall\,\mathbf{int}\ x;\ (0 \le x < \mathtt{i} \ \text{->}\ \mathtt{a}[x] \doteq 1)$

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\textbf{int}\ x;\ (0 \le x < \mathtt{a.length}\ \texttt{->}\ \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \le \mathtt{i}\ \ \&\ \ \mathtt{i} \le \mathtt{a.length}$
$\&\ \ \forall\,\textbf{int}\ x;\ (0 \le x < \mathtt{i}\ \texttt{->}\ \mathtt{a}[x] \doteq 1)$
$\&\ \ !\,\mathtt{a} \doteq \mathtt{null}$

Example

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$ & *ClassInv*

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \ \texttt{->}\ \mathtt{a[}x\mathtt{]} \doteq 1)$

Loop invariant: $0 \leq \mathtt{i}$ & $\mathtt{i} \leq \mathtt{a.length}$
& $\forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{i} \ \texttt{->}\ \mathtt{a[}x\mathtt{]} \doteq 1)$
& $!\,\mathtt{a} \doteq \mathtt{null}$
& *ClassInv'*

Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Keeping the Context

- Want to keep part of the context that is unmodified by loop

Keeping the Context

- Want to keep part of the context that is unmodified by loop
- `assignable` clauses for loops can tell what might be modified

```
@ assignable i, a[*];
```

Keeping the Context

- Want to keep part of the context that is unmodified by loop
- `assignable` clauses for loops can tell what might be modified

```
@ assignable i, a[*];
```

- How to erase all values of `assignable` locations in formula Γ ?

Keeping the Context

- Want to keep part of the context that is unmodified by loop
- `assignable` clauses for loops can tell what might be modified

```
@ assignable i, a[*];
```

- How to erase all values of `assignable` locations in formula Γ ?

  Analogous situation: $\forall$-Right quantifier rule $\Longrightarrow \forall x; \phi$
  Replace x with a fresh constant $*$

  To change value of program location use update, not substitution

Keeping the Context

- Want to keep part of the context that is unmodified by loop
- `assignable` clauses for loops can tell what might be modified

```
@ assignable i, a[*];
```

- How to erase all values of `assignable` locations in formula Γ ?

  Analogous situation: ∀-Right quantifier rule $\Longrightarrow \forall x; \phi$
  Replace x with a fresh constant $*$

  To change value of program location use update, not substitution
- Anonymising updates $\mathcal{V}$ erase information about modified locations

$$\mathcal{V} = \{\mathtt{i} := * \mid\mid \backslash\mathtt{for}\ x;\ \mathtt{a}[x] := *\}$$

Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Loop Invariants

Improved Invariant Rule

$$\Gamma \Longrightarrow \mathcal{U}[\pi\, \textbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta$$

Loop Invariants

Improved Invariant Rule

$$\Gamma \Longrightarrow \mathcal{U}\mathit{Inv}, \Delta \qquad \text{(initially valid)}$$

$$\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \texttt{p}\ \omega]\phi, \Delta$$

Loop Invariants

Improved Invariant Rule

$$\Gamma \Longrightarrow \mathcal{U} Inv, \Delta \qquad \text{(initially valid)}$$

$$\Gamma \Longrightarrow \mathcal{U}\mathcal{V}(Inv \ \& \ b \doteq \texttt{TRUE} \ \texttt{->} \ [\mathrm{p}] Inv), \Delta \qquad \text{(preserved)}$$

$$\Gamma \Longrightarrow \mathcal{U}[\pi \, \textbf{while} \ \texttt{(b)} \ \mathrm{p} \ \omega]\phi, \Delta$$

Loop Invariants

Improved Invariant Rule

$$\frac{\begin{array}{ll} \Gamma \Longrightarrow \mathcal{U}\mathit{Inv}, \Delta & \text{(initially valid)} \\ \Gamma \Longrightarrow \mathcal{U}\mathcal{V}(\mathit{Inv}\ \ \&\ \ b \doteq \texttt{TRUE}\ \ \texttt{->}\ \ [\mathrm{p}]\mathit{Inv}), \Delta & \text{(preserved)} \\ \Gamma \Longrightarrow \mathcal{U}\mathcal{V}(\mathit{Inv}\ \ \&\ \ b \doteq \texttt{FALSE}\ \ \texttt{->}\ \ [\pi\ \omega]\phi), \Delta & \text{(use case)} \end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

Loop Invariants

Improved Invariant Rule

$$\frac{\begin{array}{ll} \Gamma \Longrightarrow \mathcal{U} \mathit{Inv}, \Delta & \text{(initially valid)} \\ \Gamma \Longrightarrow \mathcal{U}\mathcal{V}(\mathit{Inv}\ \ \&\ \ b \doteq \texttt{TRUE}\ \texttt{->}\ [\mathrm{p}]\mathit{Inv}), \Delta & \text{(preserved)} \\ \Gamma \Longrightarrow \mathcal{U}\mathcal{V}(\mathit{Inv}\ \ \&\ \ b \doteq \texttt{FALSE}\ \texttt{->}\ [\pi\ \omega]\phi), \Delta & \text{(use case)} \end{array}}{\Gamma \Longrightarrow \mathcal{U}[\pi\ \mathbf{while}\ \texttt{(b)}\ \mathrm{p}\ \omega]\phi, \Delta}$$

- Context is kept as far as possible
- Invariant does not need to include unmodified locations
- For `assignable \everything` (the default):
  - $\mathcal{V} = \{* := *\}$ wipes out **all** information
  - Equivalent to basic invariant rule
  - Avoid this! Always give a specific `assignable` clause

Example with Improved Invariant Rule

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Example with Improved Invariant Rule

Precondition: $!\, \mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Example with Improved Invariant Rule

Precondition: $! \texttt{a} \doteq \texttt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\, \mathbf{int}\ x;\ (0 \leq x < \texttt{a.length} \;{-}{>}\; \texttt{a}[x] \doteq 1)$

Example with Improved Invariant Rule

Precondition: $!\,\texttt{a} \doteq \texttt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\, \textbf{int}\ x;\ (0 \leq x < \texttt{a.length} \rightarrow \texttt{a}[x] \doteq 1)$

Loop invariant: $0 \leq \texttt{i}\ \ \&\ \ \texttt{i} \leq \texttt{a.length}$

Example with Improved Invariant Rule

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\, \mathbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \mathrel{-\!\!>} \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \leq \mathtt{i}\ \ \&\ \ \mathtt{i} \leq \mathtt{a.length}$
$\&\ \ \forall\, \mathbf{int}\ x;\ (0 \leq x < \mathtt{i} \mathrel{-\!\!>} \mathtt{a}[x] \doteq 1)$

Example with Improved Invariant Rule

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \mathrel{-\!\!>} \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \leq \mathtt{i}\ \ \&\ \ \mathtt{i} \leq \mathtt{a.length}$
$\&\ \ \forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{i} \mathrel{-\!\!>} \mathtt{a}[x] \doteq 1)$

Example with Improved Invariant Rule

Precondition: $!\,\mathtt{a} \doteq \mathtt{null}$ & *ClassInv*

```
int i = 0;
while(i < a.length) {
    a[i] = 1;
    i++;
}
```

Postcondition: $\forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{a.length} \rightarrow \mathtt{a}[x] \doteq 1)$

Loop invariant: $0 \leq \mathtt{i}\ \ \&\ \ \mathtt{i} \leq \mathtt{a.length}$

$\&\ \ \forall\,\mathbf{int}\ x;\ (0 \leq x < \mathtt{i} \rightarrow \mathtt{a}[x] \doteq 1)$

Example in JML/JAVA— Demo lect14/Loop.java

```
public int[] a;
/*@ public normal_behavior
  @  ensures (\forall int x; 0<=x && x<a.length; a[x]==1);
  @  diverges true;
  @*/
public void m() {
  int i = 0;
  /*@ loop_invariant
    @  (0 <= i && i <= a.length &&
    @   (\forall int x; 0<=x && x<i; a[x]==1));
    @ assignable i, a[*];
    @*/
    while(i < a.length) {
      a[i] = 1;
      i++;
    }
  }
```

Hints

Proving `assignable`

- The invariant rule assumes that `assignable` is correct
  E.g., with `assignable \nothing;` one can prove nonsense
- Invariant rule of KeY generates proof obligation that ensures correctness of `assignable`

Hints

Proving assignable

- The invariant rule assumes that `assignable` is correct
  E.g., with `assignable \nothing;` one can prove nonsense
- Invariant rule of KeY generates proof obligation that ensures correctness of `assignable`

Setting in the KeY Prover when proving loops

- Loop treatment: Invariant
- Quantifier treatment: No Splits with Progs
- If program contains `*`, `/`:
  Arithmetic treatment: DefOps
- Is search limit high enough (time out, rule apps.)?
- When proving partial correctness, add `diverges true;`

Total Correctness

Find a decreasing integer term v (called variant)

Add the following premisses to the invariant rule:

- $v \geq 0$ is initially valid
- $v \geq 0$ is preserved by the loop body
- v is strictly decreased by the loop body

Total Correctness

Find a decreasing integer term v (called variant)

Add the following premisses to the invariant rule:

- $v \geq 0$ is initially valid
- $v \geq 0$ is preserved by the loop body
- v is strictly decreased by the loop body

Proving termination in JML/JAVA

- Remove directive `diverges true;`
- Add directive `decreasing v;` to loop invariant
- KeY creates suitable invariant rule and PO (with $\langle \ldots \rangle \phi$)

Total Correctness

Find a decreasing integer term v (called variant)

Add the following premisses to the invariant rule:

- $v \geq 0$ is initially valid
- $v \geq 0$ is preserved by the loop body
- v is strictly decreased by the loop body

Proving termination in JML/Java

- Remove directive `diverges true;`
- Add directive `decreasing v;` to loop invariant
- KeY creates suitable invariant rule and PO (with $\langle \ldots \rangle \phi$)

Example (Same loop as above)

```
@ decreasing
```

Total Correctness

Find a decreasing integer term v (called variant)

Add the following premisses to the invariant rule:

- $v \geq 0$ is initially valid
- $v \geq 0$ is preserved by the loop body
- v is strictly decreased by the loop body

Proving termination in JML/Java

- Remove directive `diverges true;`
- Add directive `decreasing v;` to loop invariant
- KeY creates suitable invariant rule and PO (with $\langle \ldots \rangle \phi$)

Example (Same loop as above)

```
@ decreasing  a.length - i;
```

Example in JML/Java— Demo lect14/LoopT.java

```
public int[] a;
/*@ public normal_behavior
  @  ensures (\forall int x; 0<=x && x<a.length; a[x]==1);
  @*/
public void m() {
  int i = 0;
  /*@ loop_invariant
    @  (0 <= i && i <= a.length &&
    @   (\forall int x; 0<=x && x<i; a[x]==1));
    @ decreasing a.length - i;
    @ assignable i, a[*];
    @*/
    while(i < a.length) {
      a[i] = 1;
      i++;
    }
  }
```

Outline

1. Motivation
2. Basic Invariant Rule
3. Anonymising Update
4. Improved Invariant Rule
5. Literature

Literature for this Lecture

Essential

KeY Book Verification of Object-Oriented Software (see course web page), Chapter 3: Dynamic Logic (Section 3.7)