Programmeren en Correctheid

Arrays

Michiel Helvensteijn (mhelvens@liacs.nl)

LIACS, Leiden University

CWI, Amsterdam

7 Mei 2010
Overview

- **Syntax**
  - March 26
  - Syntax Rules, Operational Semantics Rules, Proof Trees

- **Operational Semantics**
  - April 9
  - Axiomatic Semantics Rules

- **Axiomatic Semantics**
  - April 16
  - Weakest Preconditions, Proof Outlines

- **Arrays**
  - May 7

- **Total Correctness Rules**
  - May 21
Last Time

Last time, on P&C

• Definitions and rules for the **weakest precondition**.
Last Time

- Definitions and rules for the **weakest precondition**.
- Proof outlines!
Last time, on P&C

- Definitions and rules for the **weakest precondition**.
- Proof outlines!
- Finding the **loop** invariant.
And Now

To Do

- Syntax for arrays.
And Now

To Do

- Syntax for arrays.
- Axiomatic Semantics for arrays.
And Now

To Do

- Syntax for arrays.
- Axiomatic Semantics for arrays.
- Arrays in proof outlines.
And Now

To Do

- Syntax for arrays.
- Axiomatic Semantics for arrays.
- Arrays in proof outlines.
- Examples, examples, examples.
We’re skipping Operational Semantics for arrays!
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
- $a$ has size $|a|$, which is a natural number.
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
- $a$ has size $|a|$, which is a natural number.
- $a[1]$ is its first element. $a[|a|]$ is its last.
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
- $a$ has size $|a|$, which is a natural number.
- $a[1]$ is its first element. $a[|a|]$ is its last.
- In general, $a[A]$ denotes the element in position $z$ if $[A]_\sigma^I = z$ and $1 \leq z \leq |a|$. 

No associative arrays right now. No multi-dimensional arrays either.
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
- $a$ has size $|a|$, which is a natural number.
- $a[1]$ is its first element. $a[|a|]$ is its last.
- In general, $a[A]$ denotes the element in position $z$ if $[A]_\sigma = z$ and $1 \leq z \leq |a|$.

- No associative arrays right now.
We’re skipping Operational Semantics for arrays!

So how do these ‘arrays’ work?

- An array $a$ stores a list of integers.
- $a$ has size $|a|$, which is a natural number.
- $a[1]$ is its first element. $a[|a|]$ is its last.
- In general, $a[A]$ denotes the element in position $z$ if $[A]_\sigma^I = z$ and $1 \leq z \leq |a|$.

- No associative arrays right now.
- No multi-dimensional arrays either.
## Arrays

### Important Sets

<table>
<thead>
<tr>
<th>Set</th>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>( \mathbb{Z} )</td>
<td>{...,-2,-1,0,1,2,...}</td>
</tr>
<tr>
<td>( b )</td>
<td>( \mathbb{B} )</td>
<td>{true, false}</td>
</tr>
<tr>
<td>( x )</td>
<td>( \mathbb{V} )</td>
<td>{x, y, z, ...}</td>
</tr>
<tr>
<td>( a )</td>
<td></td>
<td>{a, b, c, ...}</td>
</tr>
<tr>
<td>( v )</td>
<td>( \mathbb{L} )</td>
<td>{i, j, k, ...}</td>
</tr>
<tr>
<td>( l )</td>
<td></td>
<td>{l, m, n, ...}</td>
</tr>
<tr>
<td>( A )</td>
<td></td>
<td>Arithmetic expressions</td>
</tr>
<tr>
<td>( A )</td>
<td></td>
<td>Extended Arithmetic expressions</td>
</tr>
<tr>
<td>( B )</td>
<td></td>
<td>Boolean expressions</td>
</tr>
<tr>
<td>( B, \phi, \psi )</td>
<td></td>
<td>Extended Boolean expressions</td>
</tr>
<tr>
<td>( C )</td>
<td></td>
<td>Commands</td>
</tr>
</tbody>
</table>
Arrays
Syntax Extension

\[ A ::= z \mid x \mid A + A \mid A \ast A \mid \alpha \mid a[A] \]
Arrays

Syntax Extension

\[
A ::= z \mid x \mid A + A \mid A \times A \mid |a| \mid a[A]
\]

\[
B ::= \text{true} \mid \text{false} \mid A < A \mid B \land B \mid \neg B \mid a = a
\]
Arrays

Syntax Extension

\[ A ::= z \mid x \mid A + A \mid A * A \mid a \mid a[A] \]

\[ B ::= \text{true} \mid \text{false} \mid A < A \mid B \land B \mid \neg B \mid a = a \]

\[ C ::= \text{skip} \mid x := A \mid a := a \mid a[A] := A \mid C ; C \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \mid \text{while } B \text{ do } C \text{ od} \]

- no operation
- assignment
- array assignment
- array indexed assignment
- sequential composition
- conditional
- while loop
Array
Assignments

\( a := b \) assigns the entire content (and size) of the array \( b \) to the array \( a \).
Array Assignments

\[ a := b \] assigns the entire content (and size) of the array \( b \) to the array \( a \).

\[ a[A] := A' \] assigns the value of \( A' \) to the position expressed by \( A \) in array \( a \).
Array
Assignments

\(a := b\) assigns the entire content (and size) of the array \(b\) to the array \(a\).

\(a[A] := A'\) assigns the value of \(A'\) to the position expressed by \(A\) in array \(a\).

\(a[A] := A'\) fails if \(\langle A, \sigma \rangle \rightarrow z\) and \((z < 1 \text{ or } |a| < z)\).
Array
Assignments

\( a := b \) assigns the entire content (and size) of the array \( b \) to the array \( a \).

\( a[A] := A' \) assigns the value of \( A' \) to the position expressed by \( A \) in array \( a \).

\( a[A] := A' \) fails if \( \langle A, \sigma \rangle \rightarrow z \) and (\( z < 1 \) or \(|a| < z \)).

But when we’re working with partial correctness, we don’t care. e.g. \( \models_{par} (true) \ a[|a| + 1] := 1 \) (\( true \)) is valid.
The rule for simple assignment of arrays is unsurprising:

\[
\begin{align*}
  \left( \phi[b/a] \right) & \quad a := b \quad \phi \\
  \quad \text{ass}
\end{align*}
\]
Indexed assignments \((a[A] := A')\) are more problematic.
Indexed assignments \((a[A] := A')\) are more problematic.

How can we substitute \(a[A]\) by \(A'\)?
Indexed assignments $a[A] := A'$ are more problematic.

How can we substitute $a[A]$ by $A'$?

- Our substitution function for formulas $\phi[y/x]$ can only take a variable name for $x$. 
Indexed assignments \( a[A] := A' \) are more problematic.

How can we substitute \( a[A] \) by \( A' \)?

- Our substitution function for formulas \( \phi[y/x] \) can only take a variable name for \( x \).
- More importantly, \( a[A] \) may have aliases in the formula. For example, \( a[3] \), \( a[1 + 2] \) and \( a[5 - 2] \) all denote the same location. \( a[x] \) too, depending on the state.
The solution

An array $a$ can be seen as a function $a : \{1, \ldots, |a|\} \rightarrow \mathbb{Z}$. 
Axiomatic Semantics of Arrays

Arrays as Functions

The solution

An array \( a \) can be seen as a function \( a : \{1, \ldots, |a|\} \rightarrow \mathbb{Z} \).

\[ a[A] \] would be the same as \( a(A) \).
Axiomatic Semantics of Arrays
Arrays as Functions

The solution
An array $a$ can be seen as a function $a : \{1, \ldots, |a|\} \rightarrow \mathbb{Z}$.

$a[A]$ would be the same as $a(A)$.

$a[A] := A'$ would be the same as $a := a[A'/A]$. 
The solution

An array $a$ can be seen as a function $a : \{1, \ldots, |a|\} \rightarrow \mathbb{Z}$.

$a[A]$ would be the same as $a(A)$.

$a[A] := A'$ would be the same as $a := a[A'/A]$.

Substitution in functions

Recall that $f[z/v_1](v_2) = \begin{cases} z & \text{if } v_1 = v_2 \\ f(v_2) & \text{if } v_1 \neq v_2 \end{cases}$
Axiomatic Semantics of Arrays

Indexed Assignment

So for our indexed assignment rule, apply the simple assignment rule to the ‘rewritten indexed assignment’:

\[
\phi[a[A'/A]/a] \quad \overline{a := a[A'/A]} \quad (\phi)
\]
So for our indexed assignment rule, apply the simple assignment rule to the ‘rewritten indexed assignment’:

\[
(\phi[a[A'/A]/a]) \quad a := a[A'/A] \quad (\phi)
\]

\textit{ass}

In summary: The Indexed Assignment rule

\[
(\phi[a[A'/A]/a]) \quad a[A] := A' \quad (\phi)
\]

\textit{iass}
So for our indexed assignment rule, apply the simple assignment rule to the ‘rewritten indexed assignment’:

$$\left( \phi[a[A'/A]/a] \right) \ a := a[A'/A] \ (\phi)$$

\textbf{ass}

In summary: The Indexed Assignment rule

$$\left( \phi[a[A'/A]/a] \right) \ a[A] := A' \ (\phi)$$

\textbf{iass}

Note that $\phi[a[A'/A]/a]$ is not the weakest precondition! It is the weakest liberal precondition. Partial correctness, for now.
Example 1

\[
\begin{align*}
&\text{true}\\
a[3] &:= 5\\
&\text{false (a[3] = 5)}
\end{align*}
\]
Example 1

\( \begin{align*} &\text{true} \\
&\left( a \left[ 5/3 \right] \left[ 3 \right] = 5 \right) \\
&a \left[ 3 \right] := 5 \\
&\left( a \left[ 3 \right] = 5 \right) \end{align*} \)
Example 1

(\textit{true})

(\texttt{a[5/3][3] = 5}) \text{ implied}

\texttt{a[3] := 5}

(\texttt{a[3] = 5}) \text{ indexed assignment}
Example 2

\(|b| > 2\)

\(a := b;\)

\(a[1] := 3;\)


\(b := a\)

\(|b[1] = 4|\)
Example 2

(| |b| > 2|)

a := b;

a[1] := 3;

(|a[1] = 4|)

b := a
(|b[1] = 4|)
Example 2

\(| |b| > 2|\)

\(a := b;\)

\(a[1] := 3;\)

\(|a[a[1] + 1/1][1] = 4|\)


\(|a[1] = 4|\)

\(b := a\)

\(|b[1] = 4|\)
Example 2

(| |b| > 2 |)

a := b;
(| a[3/1][a[3/1][1] + 1/1][1] = 4 |)

a[1] := 3;
(| a[a[1] + 1/1][1] = 4 |)

(| a[1] = 4 |)

b := a
(| b[1] = 4 |)
Example 2

```
(| \|b\| > 2 |)
(| b[3/1][b[3/1][1] + 1/1][1] = 4 |)

a := b;
(| a[3/1][a[3/1][1] + 1/1][1] = 4 |)

a[1] := 3;
(| a[a[1] + 1/1][1] = 4 |)

(| a[1] = 4 |)

b := a
(| b[1] = 4 |)
```
Example 2

\(| | b | > 2 |\)
\(| b[3/1][b[3/1][1] + 1/1][1] = 4 |\) implied

\(a := b;\)
\(| a[3/1][a[3/1][1] + 1/1][1] = 4 |\) assignment

\(a[1] := 3;\)
\(| a[a[1] + 1/1][1] = 4 |\) indexed assignment

\(| a[1] = 4 |\) indexed assignment

\(b := a\)
\(| b[1] = 4 |\) assignment
Example 3

\[
\begin{align*}
\{ a[x] = x \} \quad &
\text{assignment} \\
\{ a[a[x]] := x \} \quad &
\end{align*}
\]
Example 3

\((a[x] = x)\)

\((a[x/a[x]][x] = x)\)

\(a[a[x]] := x\)

\((a[x] = x)\)
Example 3

\[
\begin{align*}
&\{ a[x] = x \} \\
&\{ a[x/\text{a}[x]]\}[x] = x \} \quad \text{indexed assignment} \\
a[\text{a}[x]] := x \\
&\{ a[x] = x \} \quad \text{assignment}
\end{align*}
\]
Example 3

(a[x] = x)
(a[x/a[x]][x] = x) indexed assignment
a[a[x]] := x
(a[x] = x) assignment

a[x/a[x]][x] = x
Example 3

\[
\begin{align*}
\{ a[x] = x \} \\
\{ a[x/a[x]][x] = x \} & \quad \text{indexed assignment} \\
 a[a[x]] := x \\
\{ a[x] = x \} & \quad \text{assignment}
\end{align*}
\]

\[
a[x/a[x]][x] = x \iff (a[x] = x \land x = x) \lor (a[x] \neq x \land a[x] = x)
\]
Example 3

\[
\begin{align*}
&\{a[x] = x\} \\
&\{a[x/a[x]][x] = x\} \quad \text{indexed assignment} \\
&a[a[x]] := x \\
&\{a[x] = x\} \quad \text{assignment}
\end{align*}
\]

\[
a[x/a[x]][x] = x \iff (a[x] = x \land x = x) \lor (a[x] \neq x \land a[x] = x) \iff (a[x] = x \land x = x)
\]
Example 3

\[
\begin{align*}
\{ a[x] = x \} \\
\{ a[x/a[x]] [x] = x \} & \quad \text{indexed assignment} \\
\{ a[a[x]] := x \} \\
\{ a[x] = x \} & \quad \text{assignment}
\end{align*}
\]

\[
\begin{align*}
a[x/a[x]] [x] = x & \iff \\
(a[x] = x \land x = x) \lor (a[x] \neq x \land a[x] = x) & \iff \\
(a[x] = x \land x = x) & \iff \\
a[x] = x
\end{align*}
\]
We introduced simple arrays into our programming language and described their meaning.
Summary

- We introduced simple arrays into our programming language and described their meaning.
- Arrays are functions! This helps us reason about them.
Conclusion

Summary

- We introduced simple arrays into our programming language and described their meaning.
- Arrays are functions! This helps us reason about them.
- We came up with an axiomatic semantics rule for indexed assignment.
Introduction
Arrays
Axiomatic Semantics of Arrays
Examples / Conclusion

Conclusion

Summary
- We introduced simple arrays into our programming language and described their meaning.
- Arrays are functions! This helps us reason about them.
- We came up with an axiomatic semantics rule for indexed assignment.

Likely exam assignment
You will likely be asked to prove a Hoare triple for a program that uses arrays.