COMP 4161
NICTA Advanced Course
Advanced Topics in Software Verification
Gerwin Klein, June Andronick, Toby Murray

{P} ... {Q}
Invariant

Problem: While – need creativity to find right (invariant) $P$

Solution:
- annotate program with invariants
- then, Hoare rules can be applied automatically

Examples:
\[
\begin{align*}
&\{M = 0 \land N = 0\} \\
&\text{WHILE } M \neq a \text{ INV } \{N = M \ast b\} \text{ DO } N := N + b; M := M + 1 \text{ OD} \\
&\{N = a \ast b\}
\end{align*}
\]

Verification Conditions

(\{pre \ c \ Q\} \ c \ (\{Q\}) \ only \ true \ under \ certain \ conditions

These are called verification conditions $vc \ c \ Q$:

- $vc \ SKIP \ Q = True$
- $vc \ (x := a) \ Q = True$
- $vc \ (c_1; c_2) \ Q = vc \ c_2 \ Q \land (vc \ c_1 \ (pre \ c_2 \ Q))$
- $vc \ (IF \ b \ THEN \ c_1 \ ELSE \ c_2) \ Q = vc \ c_1 \ Q \land vc \ c_2 \ Q$
- $vc \ (WHILE \ b \ INV \ I \ DO \ c \ OD) \ Q = \forall \sigma. I \sigma \land \neg b \sigma \rightarrow vc \ c \ I \sigma \land \forall \sigma. I \sigma \land b \sigma \rightarrow Q \sigma \land vc \ c \ I$

$vc \ c \ Q \land (P \Rightarrow pre \ c \ Q) \Rightarrow (P \ c \ (\{Q\})$

Weakest Preconditions

\[\text{pre} \ c \ Q = \text{weakest } P \text{ such that } \{P\} \ c \ (\{Q\})\]

With annotated invariants, easy to get:

\[
\begin{align*}
\text{pre} \ SKIP \ Q & = Q \\
\text{pre} \ (x := a) \ Q & = \lambda \sigma. Q(\sigma(x := a \sigma)) \\
\text{pre} \ (c_1; c_2) \ Q & = \text{pre} \ c_1 \ (\text{pre} \ c_2 \ Q) \\
\text{pre} \ (IF \ b \ THEN \ c_1 \ ELSE \ c_2) \ Q & = \lambda \sigma. (b \rightarrow \text{pre} \ c_1 \ Q \sigma) \land \\
& \quad (\neg b \rightarrow \text{pre} \ c_2 \ Q \sigma) \\
\text{pre} \ (WHILE \ b \ INV \ I \ DO \ c \ OD) \ Q & = I
\end{align*}
\]

Syntax Tricks

\[\rightarrow x := 1 \text{ instead of } x := \lambda \sigma. 1 \text{ sucks}\]
\[\rightarrow \{\lambda \sigma. x = n\} \text{ instead of } \{x = n\} \text{ sucks as well}\]

Problem: program variables are functions, not values

Solution: distinguish program variables syntactically

Choices:
- declare program variables with each Hoare triple
  - nice, usual syntax
  - works well if you state full program and only use vcg
- separate program variables from Hoare triple (use extensible records),
  indicate usage as function syntactically
  - more syntactic overhead
  - program pieces compose nicely
Records in Isabelle

Records are a tuples with named components

Example:

```hs
record A = a :: nat
          b :: nat
```

- Selectors: `a :: A` ⇒ nat, `b :: A` ⇒ nat
- Constructors: `(a = Suc 0, b = -1)`
- Update: `r{ a := Suc 0 }`

Records are extensible:

```hs
record B = A +
          c :: nat list
```

Slide 9

Arrays

Depending on language, model arrays as functions:

- Array access = function application:
  ```hs
  a[i] = a i
  ```
- Array update = function update:
  ```hs
  a[i] := v = a := a[i:= v]
  ```

Use lists to express length:

- Array access = nth:
  ```hs
  a[i] = a ! i
  ```
- Array update = list update:
  ```hs
  a[i] := v = a := a[i:= v]
  ```
- Array length = list length:
  ```hs
  a.length = length a
  ```

Slide 10

Pointers

Choice 1

```hs
datatype ref = Ref int | Null
```
```

types heap = int ⇒ val
datatype val = Int int | Bool bool | Struct x int int bool | ...
```

- `hp :: heap, p :: ref`
- Pointer access: `p = the.Int (hp (the.addr p))`
- Pointer update: `p := v = hp := hp ((the.addr p) := v)`

- a bit klunky
- gets even worse with structs
- lots of value extraction (the.Int) in spec and program

Slide 11

Pointers

Choice 2 (Burstell '72, Bornat '00)

- struct with next pointer and element
  ```hs
  datatype next hp = Int int | Null
types elem hp = int ⇒ ref
types next hp = int ⇒ ref
```

- next :: next hp, elem :: elem hp, p :: ref
- Pointer access: `p := next` ⇒ `[the.addr p]`
- Pointer update: `p := next := v = next := next ((the.addr p) := v)`

- a separate heap for each struct field
- buys you `p := next` doesn't automatically (aliasing)
- still assumes type safe language

Slide 12
We have seen today ...
- Weakest precondition
- Verification conditions
- Example program proofs
- Arrays, pointers