Theorem Proving
Review

• Hoare Triples
  - Partial correctness: \{ P \} s \{ Q \}
  - Total correctness: [ P ] s [ Q ]

• Weakest preconditions
  - To verify \{ P \} s \{ Q \}
  - compute \textit{wp}(s, Q)\) and prove \( P \Rightarrow \text{wp}(s, Q)\)
  - hard for loops

• Verification conditions
  - like \textit{wp}, but use invariants for loops
Weakest Preconditions: Example

• \( \text{wp( } x:=x+1, \ x=y \ ) \)
  
  \[ = x+1=y \]

• \( \text{wp( } y:=y+1; x:=x+1, \ x=y \ ) \)
  
  \[ = \text{wp( } y:=y+1, \text{wp( } x:=x+1, \ x=y \ )) \]
  
  \[ = \text{wp( } y:=y+1, \ x+1=y \) \]
  
  \[ = (x+1=y+1) \]
  
  \[ = (x=y) \]
Weakest Preconditions For Loops

- \( \text{wp(while } E \text{ do } S, Q) \)
  
  \[ = \neg E \Rightarrow Q \]
  
  \( K\neg E \Rightarrow \text{wp}(S, \neg E \Rightarrow Q) \)
  
  \( K\neg E \Rightarrow \text{wp}(S, \text{wp}(while \ E \text{ do } S, Q)) \)

- \( = \neg E \Rightarrow Q \)
  
  \( K\neg E \Rightarrow \text{wp}(S, \neg E \Rightarrow Q) \)
  
  \( K\neg E \Rightarrow \text{wp}(S, \text{wp}(while \ E \text{ do } S, Q)) \)

... ad infinitum ... hard to compute
Verification Condition Generation

\[ \text{vc(while}_I \text{ E do s, Q) = } \]

\[ I \implies (E \implies \text{vc(s, I) } \land \neg E \implies Q) \]

- \( I \) holds on entry
- \( I \) is preserved in an arbitrary iteration
- \( Q \) holds when the loop terminates

- \( I \) is the loop invariant (provided externally)
- \( x_1, \ldots, x_n \) are all the variables modified in \( s \)
- Definition says:
  - the invariant holds initially,
  - and on any loop iteration where the invariant initially holds
    - if the loop terminates then the postcondition holds
    - and if the loop does not terminate, then after \( s \), the invariant holds
Weakest Precondition Generation

\[ wp(s_1; s_2, R) = wp(s_1, wp(s_2, R)) \]
\[ wp(x := E, Q) = Q[E/x] \]
\[ wp(\text{if } E \text{ then } s_1 \text{ else } s_2, Q) = (E \Rightarrow wp(s_1, Q)) \land (\neg E \Rightarrow wp(s_2, Q)) \]
\[ wp(\text{while } E \text{ do } S, Q) = \ldots \ldots \text{ hard} \]
Verification Condition Generation

\[ vc(s_1; s_2, R) = vc(s_1, vc(s_2, R)) \]
\[ vc(x := E, Q) = Q[E/x] \]
\[ vc(\text{if } E \text{ then } s_1 \text{ else } s_2, Q) = (E \implies vc(s_1, Q)) \land (\neg E \implies vc(s_2, Q)) \]
\[ vc(\text{while}_I E \text{ do } s, Q) = \]
\[ I \land (5x_1 \ldots x_n. I \implies (E \implies VC(s, I) \land 4 E \implies Q)) \]
Verification Conditions: Example

- What is $vc(\text{while}_{I} \ x > 0 \ \text{do} \ x := x - 1, \ x = 0)$
- Depends on $I$

- Suppose $I = \text{false}$
- Suppose $I = (x < 0)$
- Suppose $I = (x > 0 \ \text{and} \ x \in \text{Integer})$
VCs are less weak than WPs

- Recall what we are trying to do:

  ![Diagram with valid preconditions and verification conditions]

  - **false**
  - **true**
  - **weak**
  - **strong**

  **valid preconditions**

  - **weakest precondition**: $WP(s, Q)$
  - **verification condition**: $VC(s, Q)$
What about Exceptions?

• \( s ::= x := E \)
  
  if \( E \) then \( s \) else \( s \)

  \( s ; s \)

  while \( E \) do \( e \)

  throw

  try \( s \) catch \( s \)

• Statements may terminate \textit{normally or exceptionally}

• \( wp(s, Q, R) = \text{set of states from which} \)
  
  - \( s \) may terminate normally in a state satisfying \( Q \), or
  
  - \( s \) may terminate exceptionally in a state satisfying \( R \)
Computing WP for Exceptions

\[ wp(x := E, Q, R) = Q[E/x] \]
\[ wp(\text{if } E \text{ then } s_1 \text{ else } s_2, Q, R) = (E \Rightarrow wp(s_1, Q, R)) \land (\neg E \Rightarrow wp(s_2, Q, R)) \]
\[ wp(\text{throw}, Q, R) = wp(s_1; s_2, Q, R) = wp(s_1, Q, R) \]
\[ wp(\text{try } s_1 \text{ catch } s_2, Q, R) = wp(s_1, Q, R) \]
Computing WP for Exceptions

\[ wp(x := E, Q, R) = Q[E/x] \]
\[ wp(\text{if } E \text{ then } s_1 \text{ else } s_2, Q, R) = ( E \implies wp(s_1, Q, R)) \land (\neg E \implies wp(s_2, Q, R)) \]
\[ wp(\text{throw}, Q, R) = R \]
\[ wp(s_1; s_2, Q, R) = wp(s_1, wp(s_2, Q, R), R ) \]
\[ wp(\text{try } s_1 \text{ catch } s_2, Q, R) = wp(s_1, Q, wp(s_2, Q, R) ) \]
VCs for procedures

• Consider
  requires true
  ensures result > 0
  void abs(int x) {
    if (x<0)
      then result := -x
    else result := x
  }

• VC is
  precondition ⇒ vc( body, postcondition )
  true ⇒ vc( body, result > 0 )
  true ⇒ ((x<0 ⇒ -x>0) K y(4 (x<0 ⇒ x>0)))
ESC/Java architecture

Annotated Java program

- Sugared command
- Guarded command
- Passive command

Verification condition

- Automatic theorem prover
- Counterexample context

Post processor

Warning messages
(AND
  (<: T T |T java.lang.Object|)
  EQ T (asChild T |T java.lang.Object|))
(DISTINCT arrayType |T boolean| |T char| |T byte| |T short| |T int|
  |T long| |T float| |T double| |T .TYPE|
  T T |T java.lang.Object|)))

Verification condition

translation

Annotated Java program

Verification condition

Sugared command

Translator

Primitive command

Sugared

Passive command

Automatic theorem prover

Verification condition

Counterexample context

Warning messages

Post processor

Warning messages

Verification condition

Counterexample context

Warning messages

10 Feb 2004
Deciding Validity of VCs

• VC is
  \[\text{true} \Rightarrow ((x<0 \Rightarrow -x>0) \land (x<0 \Rightarrow x>0))\]

• Is VC valid?
• Is it true for all values of x?

• Error condition is negation of the VC
  \[\neg \left(\text{true} \Rightarrow ((x<0 \Rightarrow -x>0) \land (x<0 \Rightarrow x>0)) \right)\]

• Is EC satisfiable?
• Is EC true for any values of x?
• If so, then VC is false for that x, and so is invalid
• That x is a counter-example
Deciding Satisfiability of ECs

• A hard (but solvable) problem ...
• Start with a simpler problem ...

• Satisfiability of boolean formula (SAT)
  - canonical NP-complete problem
  - rapid progress in last few years
  - many applications for SAT solvers
    • including in theorem proving
Boolean Formulas (CNF)

- variable \( v \)
- literal \( l ::= v \mid \neg v \)
- clause \( c ::= l_1 \lor \ldots \lor l_n \)
- clause set \( s ::= c_1 \land \ldots \land c_n \)
Davis-Putnam Algorithm

• variable \( v \)
• literal \( l ::= v \mid \neg v \ldots \)
• clause \( c ::= l_1 \land \ldots \land l_n \)
• clause set \( s ::= c_1 \land \ldots \land c_n \)

• Rules
  - [Done] If a clause is empty
    • then clause set is unsatisfiable
  - [BCP] If a unit clause
    • then assign that literal true
  - [Split] Pick literal. Try assigning it true, and then try assigning its negation true
  - To assign a literal true
    • remove clauses with that literal
    • remove negation of literal from other clauses
Davis-Putnam Algorithm (cont)

- Example: (each line a separate clause)
  a b c
  a -b
  b c
  a c
  -a -b
  -c a
Next week

• Leverage SAT to decide satisfiability of ECs

• Verifun paper gives overview of this approach
  - NASA attendees: read sections 1+2 and no review.
Later today

- Arnaud Venet and Guillaume Brat on NASA's *C Global Surveyor* in Crown College, room 105, 2-3:45pm.

- Arnaud Venet
  - Rapid Inference of Interprocedural Numerical Invariants for Large C Programs by Abstract Interpretation

- Guillaume Brat
  - Software Analysis Opportunities at NASA
Presentations

- Harry: Feb 5
- Dorrit: Feb 24
- Min: March 11

- there's still room for you ...
How am I doing?