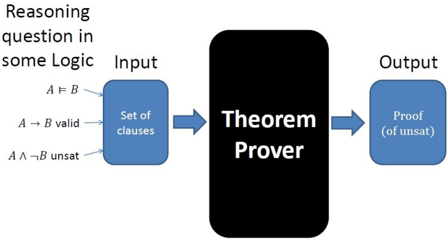
Theorem Proving

Automated Theorem Proving



Lucien Ngalamou

Outline

- A Framework for Software Verification
- Introduction to Theorem Proving (Automatic Theorem
- Hoare Logic
- Class Activity
- The KeY Project

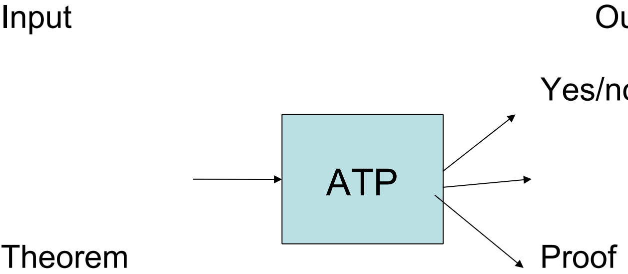
A Framework for Software Verification

- Convert the informal description R of requirem
application domain into an ”equivalent” formula
logic;

- Write a Program which is mean to realize φ_R in the
programming environment supplied by your comp
wanted by the particular customer;

- Prove that the program P satisfy the formula φ_R.

What is an automated theorem



“Countere

Example theorems

- Pythagoras theorem: Given a right tria
  sides A B and C, where C is the hypot
  $C^2 = A^2 + B^2$

- Fundamental theorem of arithmetic: An
  number bigger than 1 can be represen
  exactly one way as a product of prime

The model checking approach

- Create a model of the program in a de
  formalism
- Verify the model algorithmically
- Difficulties

- Model creation is burden on programmer – The model might be incorrect.

- If verification fails, is the problem in the … the program?

The axiomatic approach

- Add auxiliary specifications to the prog
  decompose the verification task into a
  verification tasks
- Verify each local verification problem •

- Auxiliary spec is burden on programmer –
  Auxiliary spec might be incorrect.

- If verification fails, is the problem with t
  specification or the program?

Example Theorem

- The program "z = x; z = z + y;" comput
sum of 'x' and 'y' in 'z' according to the
semantics of C

- Program-Semantics ⇒ Specification

Theorem

- Theorem must be stated in formal logi
  - self-contained
  - no hidden assumptions
- Many different kinds of logics (proposit
  logic, first order logic, higher order logi
  logic, temporal logic)

- Different from theorems as stated in m
  - theorems in math are informal
  - mathematicians find the formal details too cumbersome

Human assistance

- Some ATPs require human assistance
  - e.g.: programmer gives hints a priori, or in ATP using a prompt

- Hardest theorems to prove are “mathe
  interesting” theorems (eg: Fermat’s las
  theorem)

Output

- Can be as simple as a yes/no answer
- May include proofs and/or counterexar

- These are formal proofs, not what mathematicians refer to as proofs
- Proofs in math are
  - informal
  - “validated” by peer review
  - meant to convey a message, an intuition proof works -- for this purpose the formal too cumbersome

Output: meaning of the answer

- If the theorem prover says “yes” to a fo
  what does that tell us?
  - Soundness: theorem prover says yes imp
    is correct
  - Subject to bugs in the Trusted Computing
    (TCB)
  - Broad defn of TCB: part the system that m
    correct in order to ensure the intended gu
  - TCB may include the whole theorem prov

Output: meaning of the answer

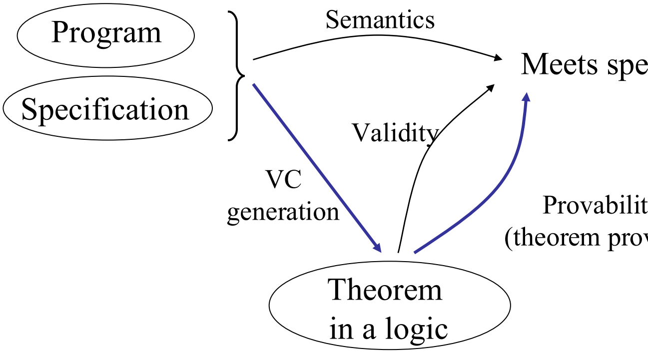
- Or it may include only a proof checker

- If the theorem prover says "no" to a for
  what does that tell us?
  - Completeness: formula is correct implies
    prover says yes
  - Or, equivalently, theorem prover says no
    formula incorrect
  - Again, as before, subject to bugs in the T

Output: meaning of the answer

- ATPs first strive for soundness, and th
  completeness if possible
- Some ATPs are incomplete: “no” answ
  provide any information
- Many subtle variants
  - refutation complete
  - complete semi-algorithm

Theorem Proving and Software


Program
Specification
Semantics
Meets spe
Validity
VC
generation
Provabilit
(theorem prov
Theorem
in a logic

- Soundness:
  - If the theorem is valid then the program m
    specification
  - If the theorem is provable then it is valid

Programs ! Theorems = Axiomatic Se

- Consists of:

- A language for making assertions about pr
- Rules for establishing when assertions hol

- Typical assertions: - During the execution
non-null pointers are dereferenced
  - This program terminates with x = 0
- Partial vs. total correctness assertions
  - Safety vs. liveness properties

- Usually focus on safety (partial correctnes

Hoare Logic

C. A. R. (Tony) Hoare

The inventor of this week's logic is also famous for inventing the **Quicksc**
algorithm in 1960 - when he was just **26**! A quote:

> Computer programming is an **exact science** in that all the propertie
> of a program and all the consequences of executing it in any given en
> vironment can, in principle, be found out from the text of the progran
> itself by means of purely **deductive reasoning**.

Hoare Logic

- A way of asserting properties of programs.
- Hoare triple: $\{A\}P\{B\}$ asserts that "If program P is

state satisfying condition A, if it terminates, it will te

- state satisfying condition B." A proof system for p
- assertions.

A way of reasoning about such assertions using th "Weakest Preconditions" (due to Dijkstra).

Example program

simple programming language skip x := *e*

(assignment) if *b* then *S* else*T* (if-

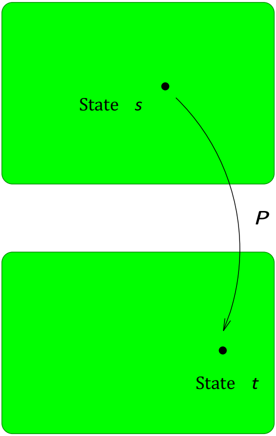
then-else) while *b* do *S* (while) *S* ; *T*

-
- (sequencing)
-
- x := n;a := 1; while (x
- ≥ 1) { a := a * x;

```
        x  := x - 1
}
```

View program P as a partial map $[P] : \mathit{Stores} \to \mathit{Stores}$.

All States

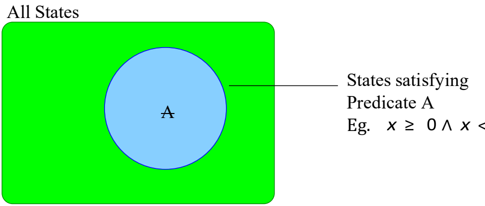


$\{x \mapsto 2, y \mapsto 10, z \mapsto$

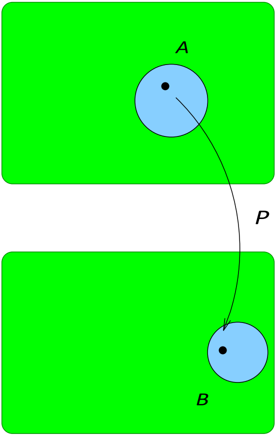
y = y + 1;z = x + y

$\{x \mapsto 2, y \mapsto 11, z \mapsto$

Predicates on States



$\{A\}P\{B\}$ asserts that "If program P is started in a state s
condition A, either it will not terminate, or it will termi



satisfying condition B."

All States

$\{10 \leq y\}$

y = y + 1;z = x + y

$\{x < z\}$

Proof rules of Hoare Logic

Skip:

$$\frac{}{\{A\}\mathrm{skip}\{A\}}$$

Assignment

$$\frac{}{\{A[e/x]\}\mathrm{x} := \mathrm{e}\{A\}}$$

If-then-else:

$$\frac{\{P \wedge b\}\ S\ \{Q\},\ \{P \wedge \neg b\}\ T\ \{Q\}}{\{P\}\text{if } b \text{ then } S \text{ else } T\ \{Q\}}$$

While (here P is called a *loop invariant*)

$$\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\text{while } b \text{ do } S\ \{P \wedge \neg b\}}$$

Sequencing:

$$\frac{\{P\}\ S\ \{Q\},\ \{Q\}\ T\ \{R\}}{\{P\}\ S\,;T\ \{R\}}$$

Class Activity

Weakening:

$$\frac{P \Longrightarrow Q,\ \{Q\}\ S\ \{R\},\ R \Longrightarrow T}{\{P\}\ S\ \{T\}}$$

- Download from BB (Week 5) and read the file hoare-
  (15 minutes to 20 minutes)
- Discussion

The KeY Project (Formal Methods for Comp Objects Conf. 2006)

- The KeY Tool (https://www.key-project.org) is a Verification of Object-Oriented Programs
- The currently most prominent applications are:
  - Program Verification (Standalone GUI, Eclipse Integr
  - Debugging (Symbolic Execution Debugger)
  - Information Flow Analysis / Security
  - Test Case generation (KeYTestGen)

Some Buzzwords Early On

- Java as target language
- Dynamic logic as program logic
- Verification = symbolic execution + induction
- Sequent style calculus + meta variables + incrementa
- Prover is interactive + automated Integration
- with two standard SWE tools:
  - TogetherCC, a commercial CASE tool
  - Eclipse, an open extensible IDE
- Specification languages
  - JML
  -
-

Supported Specification Languages: OCL

OCL/UML

Smart cards as main target application

bject Constraint Language Part

f the OMG standard UML

:ope:

dd formal constraints to UML (class) diagrams

Supported Specification Languages: JML

ıva Modeling Language

ehavioral interface specification language for Java

ıternational community effort lead by Gary T. Leavens, uilding on the Larch approach

omes with assertion and runtime checkers

CL and JML

both

- specify method behaviour: pre/post conditions
- specify admissible states: class invariants
- essentially full first order
- support inter-object navigation

differences

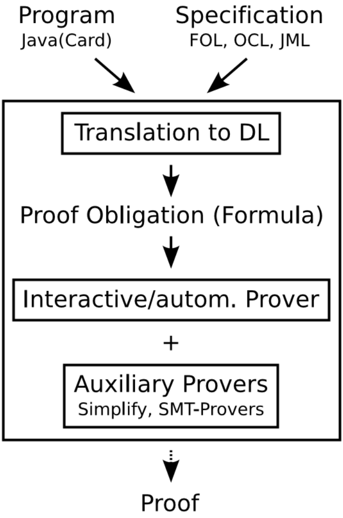
- OCL model oriented:
  - attached to class diagrams
  - ‘talks’ UML
- JML implementation oriented:
  - attached to Java programs
  - ‘talks’ Java
  - specifies exceptional behaviour also
- JML only: restricting scope of side effects

ʌL example

```
/*@ public normal_behavior
   @ requires a != null;
   @ ensures (\forall int j; j >= 0 && j < a.length;
  @                                       \result >= a[j])
   @ ensures a.length > 0 ==>
  @                              (\exists int j; j >= 0 &&
  @                                       \result == a[j])
   @*/ public static /*@ pure @*/ int max(int[] a) { if
a.length == 0 ) return 0; int max = a[0], i = 1; while ( i <
a.length ) { if ( a[i] > max ) max = a[i];
            ++i;
      }
```

```
        return max;
}
```

ɜY Architecture


Program
Java(Card)
Specification
FOL, OCL, JML
Translation to DL
Proof Obligation (Formula)
Interactive/autom. Prover
+
Auxiliary Provers
Simplify, SMT-Provers
Proof

ɔmponents of the Calculus

1. Non-program rules
   - first-order rules
   - rules for data-types (primarily: arithmetic) rules
   - for modalities
2. Rules for reducing/simplifying the program (symbolic

   Replace the program by combination of
   - case distinctions (proof branches) and
   - sequences of updates
3. Rules for handling loops
   - rules using loop invariants unwinding
   - + induction

K

Rules for replacing a method invocations by the meth

Update simplification

Deductive Verification of OO Programs: A Calculus for 100% JavaCard

The calculus covers:

- method invocation, dynamic binding
- polymorphism
- abrupt termination
- checking for nullpointer exceptions
- object creation and initialisation
- arrays
- finiteness of integer data types
- transactions (JavaCard)

By that, KeY covers the full ‘JavaCard’ language.

ɔverage of Java features

ıva Card

- Subset of Java, but with transaction concept
- Sun's official standard for Smart Cards and embe

Why Java Card?

Good example for real-world object-oriented language

JavaCard has *no*

- garbage collection
- dynamical class loading
- multi-threading
- floating-point arithmetic

Application areas

- security critical
- financial risk
  (e.g. exchanging sn
  is expensive)

ıplementing Rules: Taclets

Uniform language for different classes of rules

- First-order calculus
- Specific to JavaDL: symbolic execution for Java
- Axioms of theories: arithmetic, lists, etc.
- Lemmas

Simple, high-level language

- Adding, modifying, and removing formulas
- Conditions restricting applicability of rules
- No complex features like loops
- Suitable both for interactive and automated systems
- Lemmas are validated wrt. base taclets

brary Case Studies

va Collections Framework (JCF)

- Part of JCF (treating sets) specified using UML/OCL
- parts of reference implementation verified

va Card API

- Most parts of JavaCard API specified using UML/O
- parts of reference implementation verified

:horr-Waite Algorithm

- Standard benchmark for verification systems
- Graph marking algorithm for garbage collection
-
-
-

K

Java implementation: 2 classes, core algorithm 25 line

Heavy aliasing, frame problem

Specified and verified

ecurity Case Studies: Java Card Software

afety/security properties specified in dynamic logic

- 'Only certain exceptions can be thrown'
- Transactions are properly used
  (do not commit or abort a transaction that was neve
  started Transactions are also closed)
- Data consistency
  (also if a smartcard is "ripped out" during operation)
- of overflows for integer operations

-

Demoney (about 3000 lines):

Electronic purse application provided by Trusted Logic

SafeApplet (about 600 lines): RSA based authenticatio

afety Case Study

Software by DBSystems for computing schedules for t

Speed restrictions, required break powers

Software formally specified using UML/OCL (based on existing informal specification)

-
-
- Program translated from Smalltalk to Java

vionics Software

- Java implementation of a Flight Manager module at T
- Comprehensive specification using JML, emphasis or
- Verification of some nested method calls using contra

rtual Machine for Real Time Secury Java

- Verification of some library functions of the Jamaica V