Theorem Prover:
Prototype Verification System

Thanks to Prof. Andrea Domenici for providing useful inputs for this slides

Formal system

A formal system is a system that we use to prove the truth of sentences by deductions, i.e., by showing that a sentence follows through a series of reasoning steps from some other sentences that are known (or assumed) to be valid.

A formal system consists of:

- A set of axioms, selected sentences taken as valid.
- A set of inference rules, saying that a sentence of a given structure can be deduced from sentences of the appropriate structure, independently of the meaning (semantics) of the sentences.
  - E.g., A and B stand for any two sentences, a well-known inference rule says that from “A” and “A implies B” we can deduce B.

Elements of a formal system

We have a formal system **F** with the set of axioms **A** and the set of inference rules **R**

We want to prove that a formula **s** follows from a set **H** of hypotheses.

A deduction of **s** from **H** within **F** is a sequence of formulae such that **s** is the last one and each other formula either:

1. belongs to **A**; or
2. belongs to **H**; or
3. is obtained by applying some rules belonging to **R** to some preceding formulae

Example of Formal system

- **A**: $\{\forall x \in \mathrm{R}: x^0 = 1\}$
- **R:** $\{\forall x,y \in R: (x=y) \Rightarrow (x - y = 0),$

  $\forall x \in \mathrm{R}: (a = x)$ AND $(b = x) \Rightarrow (a = b),$

  $(a = b) \Rightarrow (s(a) \Rightarrow s(b))\}$

- **H**: $\{a = b\}$
- **s**: $x_1^{(a-b)} = y_1^{(a-b)}$

1. $a = b$
2. $a - b = 0$
3. $x_1^0 = 1$
4. $y_1^0 = 1$
5. $x_1^0 = y_1^0$
6. $x_1^{(a-b)} = y_1^{(a-b)}$

Theorem Proving

- A theorem prover is a computer program that implements a formal system.
- It takes as input a formal definition
  - of the system that must be verified (**H**)
  - of the properties that must be proved (**s**),
- and tries to build a proof by application of inference rules, in an automatic or semi-automatic way.

- Generally speaking a theorem prover provides the base set **R** of inference rules along with the base set **A** of the axioms.
  - Users provide **H** and **s**

Prototype Verification System

- The PVS is an interactive theorem prover developed at Computer Science Laboratory, SRI International, Menlo Park (California), by S. Owre, N. Shankar, J. Rushby, and others.

- The formal system of PVS consists of a language and the sequent calculus axioms and inference rules.

- PVS has many applications, including formal verification of hardware, algorithms, real-time and safety-critical CPS.

Sequent calculus

- The sequent calculus works on sentences called sequents, of this form:

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$$

where the A's and B's are the antecedents and the consequents, respectively.

- The symbol in the middle ($\vdash$) is called a turnstile and may be read as "yields".

- A sequent can then be seen informally as another notation for

$$A_1 \land A_2 \land \ldots \land A_n \Rightarrow B_1 \lor B_2 \lor \ldots \lor B_m$$

Application of sequent calculus

Proofs are constructed backwards from the goal sequent, which has the form

$$\mathbf{H \vdash s}$$

where **s** is the formula we want to prove and H are our hypothesis.

Inference rules are applied backwards, i.e., given a formula, we find a rule whose consequence matches the formula, and the premises become the new subgoals.

Since a rule may have two premises, proving a goal produces a tree of sequents, rooted in the goal, called the proof tree.

The proof is completed when (and if!) all branches terminate with a true sequent

Proved sequence

A sequent is proved(true) if:

1. any antecedent is false; or
2. any consequent is true;
3. any formula occurs both as an antecedent and as a consequent.

x	y	x => y
False	False	True
False	True	True
True	False	False
True	True	True

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee B_2 \vee \ldots \vee B_m$$

PVS Theories

- A PVS specification is composed of one or more theories.

```
<name>: THEORY
    BEGIN
        <imports>
        <type declarations>
        <constant and functions declarations>
        <formulae>
    END <name>
```

- Constructs of different classes may be interleaved (e.g., a type declaration may follow a variable declaration), but every symbol must be declared before it is used.

The PVS Specification Language

- Logical connectives: NOT, AND, OR, IMPLIES, . . .
- Quantifiers: EXISTS, FORALL.
- Complex operators: IF-THEN-ELSE, COND.
- Notation for records (i.e. C struct type). . .
- Theories: named collections of definitions and formulae. A theory may be imported(and referred to) by another theory.
- A large number of pre-defined theories is available in the **prelude** library.