# Introduction to PVS

Dawn Song
UC Berkeley

# Current Thoerem Provers (Backward chaining)

- HOL

  http://www.cl.cam.ac.uk/Research/HVG/HOL/

  developed in Europe, in ML

- Isabelle

  http://www.cl.cam.ac.uk/Research/HVG/Isabelle/index.html

  Cambridge University (Larry Paulson) and TU Munich (Tobias Nipkow), in ML

  E.g. security protocol analysis

- ACL2

  http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html

  UT Austin, Matt Kaufmann and J Strother Moore, in Lisp

  E.g. validation of Java bytecode verifier

# History of PVS

- Prototype/People's Verification System
- By Sam Owre, N. Shankar, John Rushby with help at CSL SRI, http://www.csl.sri.com
- Builds on near 20 years experience and many years of coding
- Freely available on platforms including Linux, SunOS, written in Allegro Lisp

# Role of such a tool in reality

- Not just to prove mathematical theorems, but rather to provide high assurance to critical systems
  - Specification language for formalizing designs of systems and protocols
  - Proof engine for property vefication
  - Can also debug designs, derive test cases, adjust alternative design
- Stages:
  - Write down specification of systems and specification of properties
  - Interactive proof with PVS's automation support
  - Usually takes weeks or months of human effort
  - Gap between spec and implementation

# Applications I

- Hardware verification:
  - verification of avionics processor AAMP5 (on board Boeing 747)
  - hierarchical verification of two-dimensional high-speed multiplication circuit
- Protocol verification:
  - Communication protocols: fiber chanel protocol verification
  - Cache Coherence Protocols and Memory Models
- Requirements validation and software verification:
  - Formalizing New Navigation Requirements for NASA's Space Shuttle, and specification of a simple auto-pilot
  - Cassini spacecraft (a Saturn orbiter due for launch in 1997)
  - High-level design and validation of ATM Switches

# Applications II

- Compiler verification: e.g. security of Java-style dynamic linking
- Security analysis:
  - Security Protocol Analysis
  - A Logic of Composition for Information Flow Predicates
- OS verification: e.g. formal analysis of a real-time kernel specification
- Verifying the IO and timed automata of Nancy Lynch
- Fault-tolerance guarantee: e.g. fault-tolerant clock synchronization circuit

## PVS Overview: a general purpose verification system

- An expressive specification language, based on classical higher-order logic with paramertized theories
- An interactive theorem prover with powerful automation
- Emacs interface and extensive browsing and reporting capabilities
- Latex-prettyprinter

# Basics of PVS proof system

1. Specification is a high-order logic with parameterized theories, type-constructors, predicate subtypes, recursive functions, abstract data types
2. Undecidable typechecking – TCC (type-correctness conditions)
3. Built-in “prelude” and loadable “libraries” of domain-specific theories (e.g. arithmetic)
4. Many proof commands and proof strategy (e.g. grind)
5. Uses a sequent calculus presentation: sequent: conjunction of antecedent imply disjunction of consequent

## how to use PVS

1. C-c h: pvs-help
2. C-x C-f sum.pvs:

   recursive function.
3. M-x pa
4. C-c C-t: TCC: type-correctness conditions
   (a) C-c C-q s: show tccs
   (b) M-x tcp: discharge automatically
5. C-c p
6. (induct "n")
7. (postpone)
8. (expand "sum") : expand the def of sum

9. (assert)
10. (skolem!) : skolem with arbitrary constant names
11. (flatten) : simplify the formula
12. (expand ”sum” +) : expand def of sum in the consequent
13. (assert) : invoke the PVS decision procedure (arithmetic and equality procedures) which automatically decide certain fragments of arithmetic
14. M-x show-last-proof : show the .prf, the whole proof
15. M-x spt : show the status including TCCs and the thm
16. M-x spc : show dependencies