ACL2

Interactive Theorem Proving

Michael Plattner

Introduction

What is ACL2

ACL2 is a logic and programming language in which you can model computer systems, together with a tool to help you prove properties of those models. "ACL2" denotes "A Computational Logic for Applicative Common Lisp".

Common Lisp

- logic of ACL2 is based on Common Lisp
- Common Lisp is the standard list processing programming language
- ACL2 formalizes only a subset of Common Lisp

Applications

Formal Verification

- tools to *formally verify* hardware and software systems
- augmenting traditional testing with proof
- interactive theorem provers

Organisation

- IBM - floating point divide and square root
- AMD - verify floating point operations with IEEE 754
- Sun Java Virtual Machine - bytecode verifier

Logic of ACL2

Mathematical Logic

- formal system of formulas (axioms) and rules
- deriving theorems
- proof is a derivation of a theorem with a proof tree

According to ACL2, some strengths among ITPs

- proof automation
- proof debugging utilities
- Fast execution
- Documentation

Basic ACL2 Demo

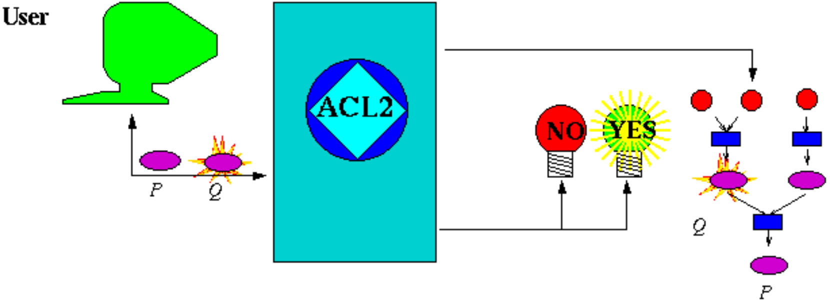
`(+ 3 4)`	`7`
`(defun f (x)` `   (+ x 10))`	`Since F is non-recursive,` `its admission is trivial....`
`(f 3)`	`13`
`(* (f 0) (f 1))`	`110`
`(cons 1 (cons 2 nil))`	`(1 2)`
`'(1 2)`	`(1 2)`
`(consp '(1 2))`	`T`
`(car '(1 2))`	`1`
`(cdr '(1 2))`	`(2)`

ACL2 Demo

```
(defun app (x y)
(if (consp x)
    (cons (car x) (app (cdr x) y))
  y))
(app '(1 2) '(a b c))
(app '(1 2)
   (app '(a b c) '(4 5)))
(app (app '(1 2) '(a b c))
   '(4 5))
(thm
(equal (app (app x y) z)
      (app x (app y z))))
```

```
The admission of APP is trivial,
using the relation O< ...



(1 2 A B C)
(1 2 A B C 4 5)

(1 2 A B C 4 5)

*1 (the initial Goal, a key checkpoint)
```

Guiding proofs



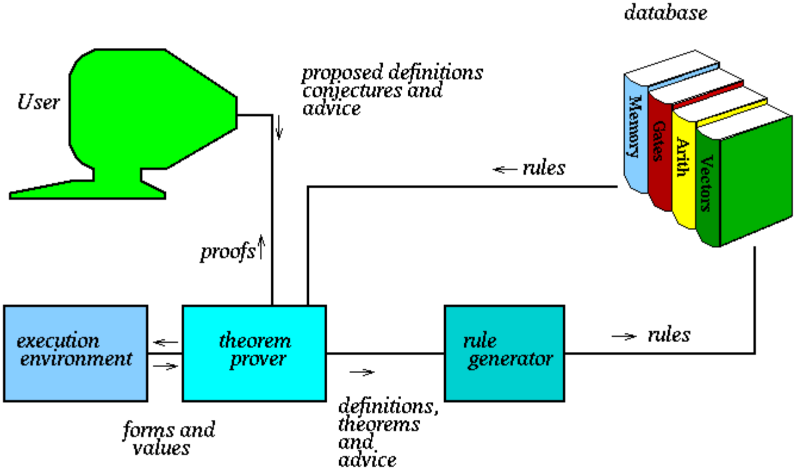
- Q is important lemma to prove P
- user first proves Q
- Q is found by failed prove of P

Rules

Theorem

- proven theorem converted into one or more rules
- stored in a database
- proving theorems leads to control over ACL2s automation
- rewrite rule is most common rule

ACL2 System architecture



Demo 2

```
(include-book "arithmetic/top" ; include a "community book"
:dir :system)
(defthm sum-to-n-rewrite ; Prove and store a rewrite rule
                 ; to replace sum-to-n(n) by n(n+1)/2.
(implies (natp n)
   (equal (sum-to-n n)
          (/ (* n (+ n 1))
             2))))
(thm ; proof succeeds immediately
(implies (natp k)
         (equal (sum-to-n (* 2 k))
               (* k (+ (* 2 k) 1)))))
```

Logical Foundations

Logic

- first-order logic with induction
- ACL2 theories extend a given ground-zero theory
- peano arithmetic with $\epsilon - 0$ induction
- extended with data types

five common Lisp datatypes

- the precisely represented, unbounded numbers (integers, rationals, and the complex numbers with rational components)
- the characters with ASCII codes between 0 and 255
- strings of such characters
- symbols (including packages)
- conses (closure under a pairing operation)

Evolving Theories

Conservative extensions

Suppose theory T_1 extends theory T_0 . Then T_1 is a *conservative extension* of theory T_0 if every theorem of T_1 in the language of T_0 is a theorem of T_0 .

ACL2 extensions

- ACL2 extensions are by definition conservative
- even recursive definitions, because termination has to be proven

New concepts

- sometimes new concepts for proofs needed
- also program verification may need additional concepts
- must be done conservatively in order to believe results

Thank you for your attention!

Michael Plattner

Formal Verification

Translator

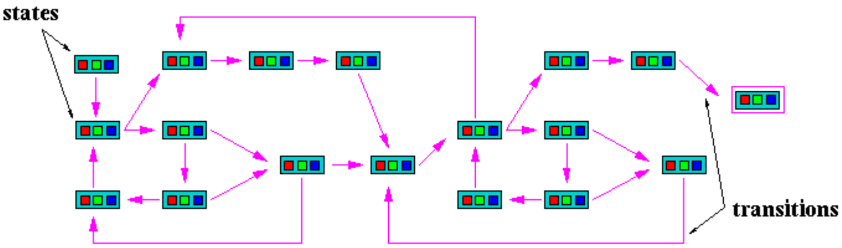
- using a translator $\rightarrow$ map programs to acl2 functions
- called *shallow embedding*

Interpreter

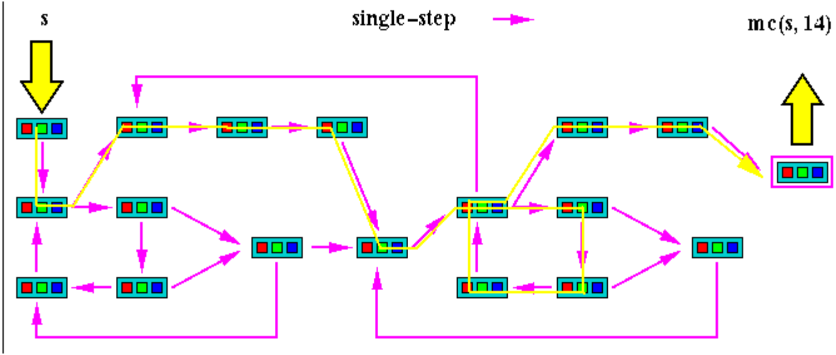
- run instruction for a number of calls
- called *deep embedding*

```
(defun mc (s n)
(if (zp n) ; n is 0
s (mc (single-step s) (- n 1)))) ; run one instruction
```

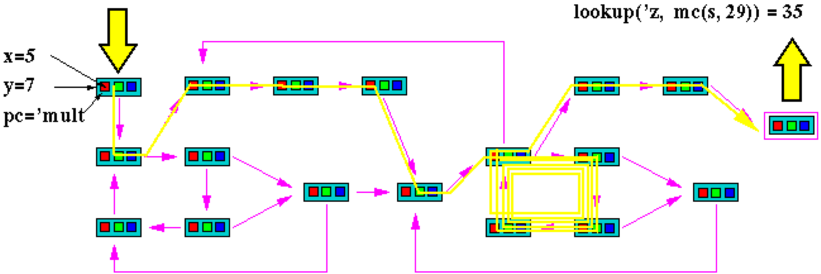
Hardware verification



Hardware verification

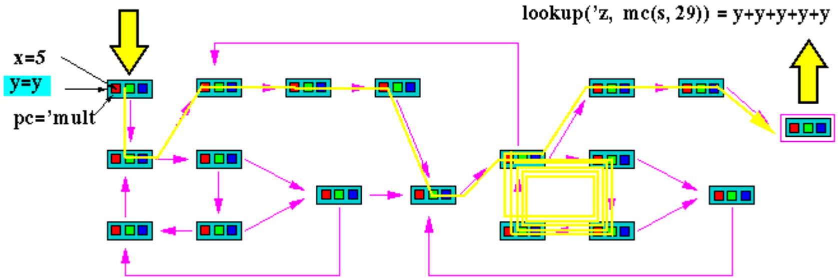


Hardware verification



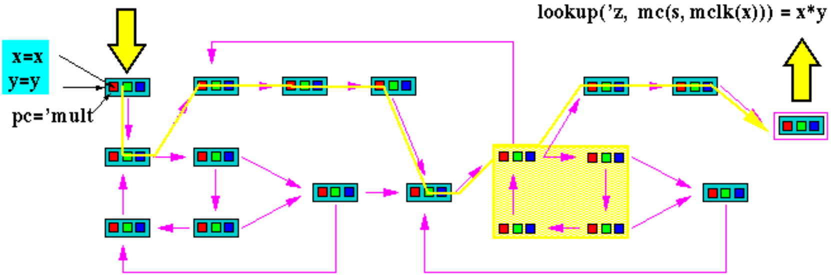

```
(lookup 'z (mc (s 'mult 5 7) 29)) ; ACL2 computes 35
```

Hardware verification




```
(lookup 'z (mc (s 'mult 5 y) 29)) ; we get (+ y y y y y).
```

Hardware verification




```
Theorem.  MC 'mult is a multiplier
(implies (and (natp x)
         (natp y))
    (equal (lookup 'z (mc (s 'mult x y) (mclk x)))
           (* x y))).
```

- ACL2 Online Manual
  Link
- ACL2 Intro
  Link
- Implementation of a Computational Logic
  Link