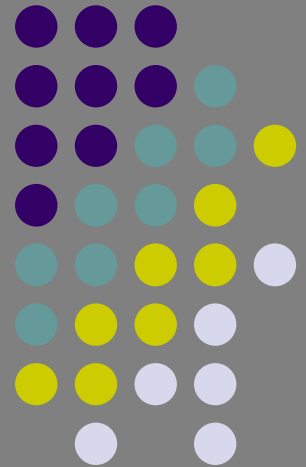
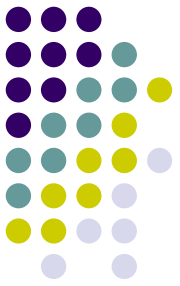


ACL2 Theorem Prover

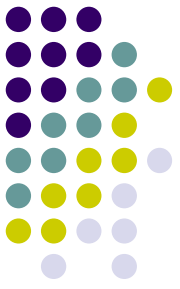
By Gabe Shelley and Steve
Forrest





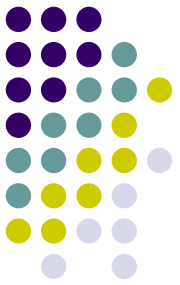
Outline

- Introduction
- Logic Overview
- Extension Principles
- Practical Theorem Proving
- Questions
- References



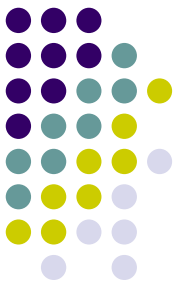
Introduction

- ACL2 stands for A Computation Logic for Applicative common LISP (developed by Boyer, Kaufmann and Moore)
 - Decendent of Nqthm (Boyer-Moore Logic)
 - Not scalable
 - All function definitions must be “proven”
 - Consists of: functional programming language (Common LISP), first order logic and mechanical theorem prover
 - Subset of LISP (side-effect free) and FOL (Quantifier free + recursive functions)



Introduction

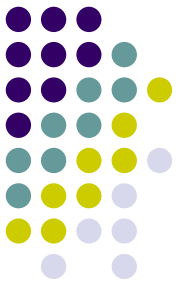
- ACL2 UI
 - Default
 - text-based
 - Interact via a read-val-print loop
 - DrACuLa
 - Plugin for Dr.Scheme (Common GUI environment for scheme and lisp based programming languages)



Logic Overview

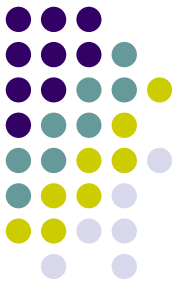
- Recursively defined total functions
 - All ACL2 functions are total. Functions which correspond to “mathematically partial” functions (e.g. $1/x$, $\text{sqrt}(x)$) are total, but their mathematically undefined arguments are handled via guards.
 - Partial function macro have been created by Manolios and Moore
- First-order logic with limited support for quantification
 - `defun-sk` provides “syntactic sugar” for quantifiers by translating quantifiers into well-founded recursion.
- Untyped (Uni-type)
 - enforced by guards or checks in function definitions
 - i.e. `(natp x)`, `(rationalp x)`
- Measure:
 - Every function has a ‘measure’ defined for its input. Arguments to recursive calls must have strictly decreasing measure.
- Loop-stopper
 - Ensures that the term-rewriting in ACL2’s proof system is confluent.

Extension Principles

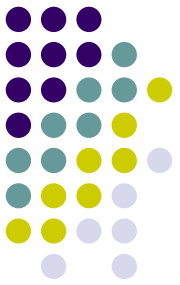


- Definition
 - In general, functions must be proven to terminate and theorems must be proven correct. However, using the “skip-proof” tag in definitions waives this requirement
- Encapsulation
 - “Book” definitions of functions, theorems and references to other books
 - Information hiding of lemma details while maintaining rules in database
 - Syntactic checks are made when multiple books are incorporated into one database to detect name collisions and ensure logical compatibility.

Practical Theorem Proving



- Proofs are generated by supplying hints within a theorem definition to help guide the theorem prover
 - Hints are essentially additional instructions applied to a particular subgoal
- Almost entirely automated.
 - Define everything before hand and let the system “run”
- Since all definitions are recursive, induction proofs are the most common type
- Proof trees are generated for a particular theorem

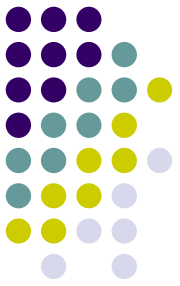


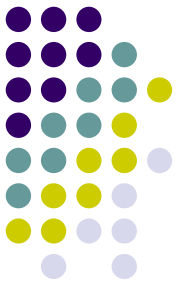
Example

- Here we demonstrate an ACL2 theorem proven with a customized induction scheme.
- ```
(defun even-induction (x)
 "Induct with increment of 2"
 (if (or (zp x) (equal x 1))
 x
 (1+ (even-induction (1- (1- x)))))))
```
- ```
(defthm even-square-has-even-base
  (implies (and (integerp p)
                (<= 0 p) (evenp (* p p)))
           (evenp p))
  :hints ((("Goal"
            :induct (even-induction p)))
          :rule-classes nil)
```


Questions

- Questions or comments?





References

- Kaufmann, M., Moore, J. – ‘*Design Goals for ACL2*’,
<http://citeseer.ist.psu.edu/cache/papers/cs/284/ftp:zS>
- Kaufmann, M., Moore, J. – ‘*A Precise Description of the ACL2 Logic*’,
<http://citeseer.ist.psu.edu/cache/papers/cs/1068/http:>