Support for “Trusted” Extension in ACL2

J Strother Moore
(joint work with Matt Kaufmann)

Department of Computer Sciences
University of Texas at Austin

August, 2010
A Computational Logic for Applicative Common Lisp = ACL2

- a functional programming language
- a first-order mathematical theory
- a mechanized theorem prover
- implemented primarily in ACL2
Primary Concerns

• soundness

• industrial-scale usability

Our primary “customers” are AMD, Rockwell-Collins, Centaur Technology, IBM, and various government agencies
We must adhere to Common Lisp
We must adhere to Common Lisp . . .
because efficient execution of ACL2 models is a major (driving?) concern
Soundness is based on the care Kaufmann and Moore have taken in the implementation

ACL2 is not “foundational” – we strive for good design and elegance in our coding, but we are willing to add logically “redundant” features as necessary
“Blessed” extension mechanisms are primarily based on proof of appropriate properties.

Our “trust story” is that if users stick with certain features, they preserve as much soundness as we had in the first place.

Users can always go “under the hood” and do anything in Lisp.
Keys to ACL2’s extensibility include

- expressions “are” objects
- user can access the state of the system
- system is coded in ACL2 so system functions are available in many contexts
Two Senses of “Extension”

● Logical – changing the logical theory

● Behavioral – changing the behavior of the prover
Logical Extension Facilities

- Ground-zero theory (starting point)
- Theory Extension Events
  - Simple axiomatic events
    - DEFUN - intro new rec fns; conservative
    - DEFCHOOSE (basis for DEFUN-SK) - witness fns; conservative
    - DEFAXIOM - risky; rarely used
  - Non-axiomatic events: DEPTHM - prove a theorem
  - Compound
    - PROGN - grouping
    - LOCAL - scoping
    - INCLUDE-BOOK - import pre-certified events
    - ENCAPSULATE - intro constrained un-interp fns
- Syntax extensions
  - DEFCONST - abbrev constants
  - DEFMACRO - computed trans of new syntax
DEMO: Logical Extension Facilities

- Ground-zero theory (starting point)
- Theory Extension Events
  - Simple axiomatic events
    - DEFUN – intro new rec fns; conservative
    - DEFCHOOSE (basis for DEFUN-SK) – witness fns; conservative
    - DEFAXIOM – risky; rarely used
  - Non-axiomatic events: DEFTHM – prove a theorem
  - Compound
    - PROGN – grouping
    - LOCAL – scoping
    - INCLUDE-BOOK – import pre-certified events
    - ENCAPSULATE – intro constrained un-interp fns
- Syntax extensions
  - DEFCONST – abbrev constants
  - DEFMACRO – computed trans of new syntax
Two Senses of “Extension”

- Logical – changing the logical theory
- Behavioral – changing the behavior of the prover
Irrelevance
User Equality
Destructor Elimination
Generalization
Induction

Simplification
evaluation
propositional calculus
BDDs
equality
uninterpreted function symbols
rational linear arithmetic
rewrite rules
recursive definitions
back- and forward-chaining
metafunctions
congruence-based rewriting
key lemma

axiom

rule of inference

theorem

main theorem
Q.E.D.
of "books" of definitions, theorems, and advice

User

proposed definitions, conjectures, and advice

proofs

Memory

Gates

Arith

Vectors

data base composed

Q.E.D.
database composed of “books” of definitions, theorems, and advice

proposed definitions, conjectures, and advice

proofs

User

Q.E.D.

theorem prover

Memory

Gates

Arith

Vectors
database composed of "books" of definitions, theorems, and advice

User

proposed definitions conjectures and advice

proofs

theorem prover

Q.E.D.
database composed of "books" of definitions, theorems, and advice
Behavorial Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
    - Pragmas -- syntaxp, bind-free, force, case-split, double-rewr
  - Static (Goal Specific) Hints
- Programmatic (analogous to tactics)
  - Computed Hints
  - Make-event
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  -
- Verified extensions
  -
- Using ACL2 as a System-Building Shell
DEMO: Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
    - Pragmas -- syntaxp, bind-free, force, case-split, double-rewrite
  - Static (Goal Specific) Hints
- Programmatic (analogous to tactics)
  - Computed Hints
  - Make-event
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  ...
- Verified extensions
  ...
- Using ACL2 as a System-Building Shell
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Pragmas -- syntaxp, bind-free, force, case-split, double-rewr
  - Static (Goal Specific) Hints
- Programmatic (analogous to tactics)
  - Computed Hints
  - Make-event
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  ...
- Verified extensions
  ...
- Using ACL2 as a System-Building Shell
Clause Processors
Verified clause processors are like metafunctions except operate at the goal level rather than the subterm level.

Unverified clause processors are external tools (like SAT-solvers, IBM’s SixthSense, etc.)
It is possible to introduce partially constrained functions whose execution is carried out by calls to external tools.

DEMO: Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
    - Pragmas -- syntaxp, bind-free, force, case-split, double-rewrite
  - Static (Goal Specific) Hints
- Programmatic (analogous to tactics)
  - Computed Hints
  - Make-event
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  -
- Verified extensions
  -
- Using ACL2 as a System-Building Shell
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Pragmas -- syntaxp, bind-free, force, case-split, double-rewrite
- Static (Goal Specific) Hints
- Programmatic (analogous to tactics)
  - Computed Hints
  - Make-event
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  ... 
- Verified extensions
  ... 
- Using ACL2 as a System-Building Shell
(defp run (s)
  (if (haltedp s)
      s
      (run (step s))))

defp ("define partial function") book: establishes that generic (uninterpreted) tail-recursive equation is satisfiable by an admissible function and then functionally instantiates that result for the user’s fns
Behavioral Extension Facilities

• Customization of Built-in Features
  ○ Extending automation through rule-classes
    • Rewriting (conditional, contextual, congruence-based)
    • Metafunctions and Verified Clause-Processors
  ○ Hints -- Static (Goal Specific) and/or Computed

• Programmatic (analogous to tactics)
  ○ Macros to generate events -- e.g., support for partial functions

• Extending evaluation capabilities:
  ○ Prototype without proof -- e.g., program mode, skip-proofs
  ○ Optimizing Evaluation -- guard, mbe

• Unverified (but useful) extensions
  ...

• Verified extensions
  ...

• Using ACL2 as a System-Building Shell
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Hints -- Static (Goal Specific) and/or Computed
- Programmatic (analogous to tactics)
  - Macros to generate events -- e.g., support for partial functions
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  - Trust tags (used at Centaur and in ACL2s)
  - Feature-based
    - Hash-cons, memoization, applicative hash tables
    - ACL2(r)
    - Parallel ACL2
- Verified extensions
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Hints -- Static (Goal Specific) and/or Computed
- Programmatic (analogous to tactics)
  - Macros to generate events -- e.g., support for partial functions
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  ...
- Verified extensions
  ...
- Using ACL2 as a System-Building Shell
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Hints -- Static (Goal Specific) and/or Computed
- Programmatic (analogous to tactics)
  - Macros to generate events -- e.g., support for partial functions
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  - ...
- Verified extensions
  - Defattach (also for testing) -- contributed talk
  - Untranslate and untranslate-preprocess
  - Verified clause-processors
Behavioral Extension Facilities

- Customization of Built-in Features
  - Extending automation through rule-classes
    - Rewriting (conditional, contextual, congruence-based)
    - Metafunctions and Verified Clause-Processors
  - Hints -- Static (Goal Specific) and/or Computed
- Programmatic (analogous to tactics)
  - Macros to generate events -- e.g., support for partial functions
- Extending evaluation capabilities:
  - Prototype without proof -- e.g., program mode, skip-proofs
  - Optimizing Evaluation -- guard, mbe
- Unverified (but useful) extensions
  ...
- Verified extensions
  ...
- Using ACL2 as a System-Building Shell
Questions?