Lecture 7

Pete Manolios
Northeastern

Professional Method

```
(definec app (a :tl b :tl) :tl
  (if (endp a)
       b
     (cons (car a)
           (app (cdr a) b))))
```

```
(definec rev (x :tl) :tl
  (if (endp x)
       nil
     (app (rev (cdr x))
          (list (car x)))))
```

Prove: (rev (rev x)) = x No quite right, why?

Prove: (tlp x) ⇒ (rev (rev x)) = x Contract completion!

Professional Method: use abbreviations, discover induction scheme

We'll induct on (... x). Base case is trivial, so go to induction step

```
          (R (R x))
= {Def R} (R (A (R (cdr x)) (L (car x))))
= {L1}    (A (R (L (car x))) (R (R (cdr x))))
= {IH}    (A (R (L (car x))) (cdr x))
= {Def R} (A (L (car x)) (cdr x))
= {Def A} x
```

Hm, to use IH, need lemma

Now I can use IH

Just equational reasoning

L1.(R (A x y)) = (A (R y) (R x))

What Induction scheme?

(tlp x) or (rev x): minor differences

Professional Method

```
(definec app (a :tl b :tl) :tl
  (if (endp a)
      b
    (cons (car a)
          (app (cdr a) b))))
```

```
(definec rev (x :tl) :tl
  (if (endp x)
      nil
    (app (rev (cdr x))
         (list (car x)))))
```

Prove: (tlp x) ∧ (tlp y) ⇒ (R (A x y)) = (A (R y) (R x))

Professional Method: induct on? x controls both LHS, RHS, so probably x

Start with induction step

```
          (R (A x y))
= {Def A} (R (cons (car x) (A (cdr x) y)))
= {Def R} (A (R (A (cdr x) y)) (L (car x)))
= {IH}    (A (A (R y) (R (cdr x))) (L (car x)))
= {Ass A} (A (R y) (A (R (cdr x)) (L (car x))))
= {Def R} (A (R y) (R x))
```

Ass A: (A (A x y) z) = (A x (A y z))
What Induction scheme?
(tlp x) or (rev x): minor differences

Base case?

```
          (R (A x y))
= {Def A} (R y)

          (A (R y) (R x))
= {Def R} (A (R y) nil)
= {L2!}   (R y)
```

L2: (A x nil) = x
Needs proof by induction!

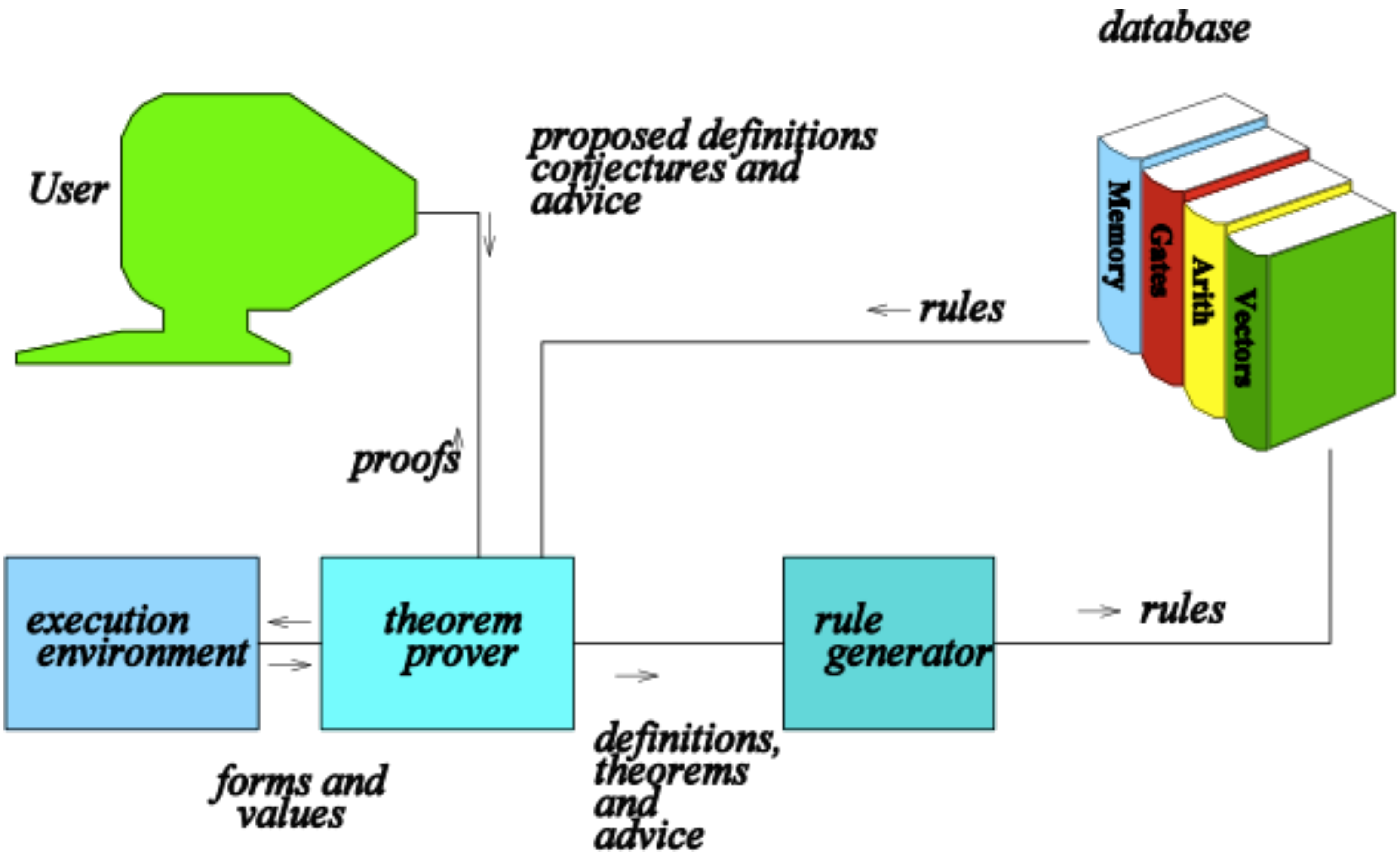
Slides by Pete Manolios for CS4820

ACL2 is . . .

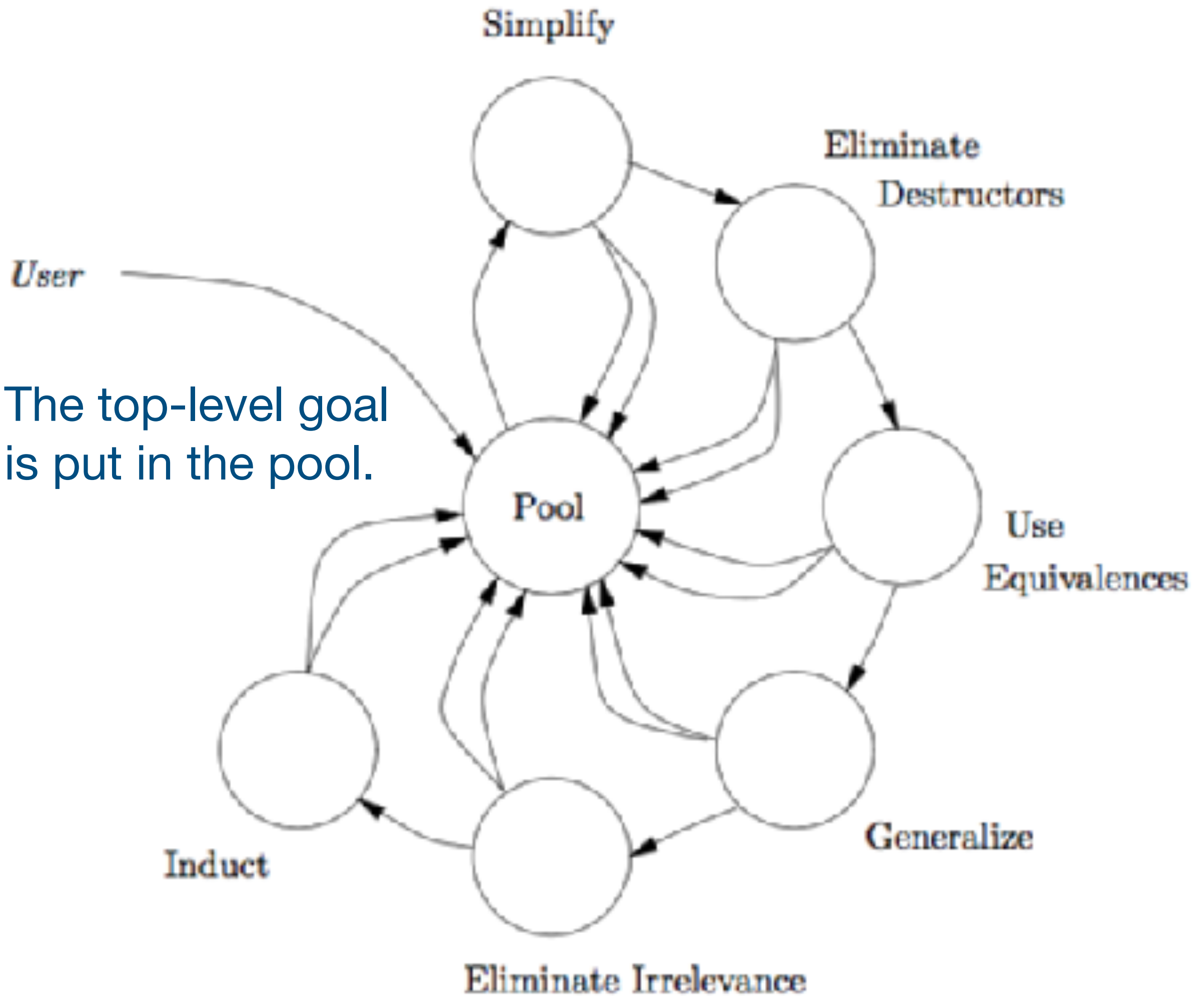
- **A programming language:**
  - Applicative, functional subset of Lisp
  - Compilable and executable
  - Untyped, first-order
- **A mathematical logic:**
  - First-order predicate calculus
  - With equality, induction, recursive definitions
  - Ordinals up to ϵ_0 (termination & induction)
- ***A mechanical theorem prover:***
  - *Integrated system of ad hoc proof techniques*
  - *Heavy use of term rewriting*
  - *Largely written in ACL2*

Slides by Pete Manolios for CS4820

ACL2 System Architecture



Organization of ACL2



The top-level goal is put in the pool.

When a formula is drawn out, it is passed to proof techniques until one applies.

The draw is orchestrated that we do not try to prove a subgoal by induction until we have processed every subgoal produced by the last induction.

Induction

- When a formula arrives at the induction technique, ACL2 computes all the inductions suggested by the terms in the formula.
- It then compares them, possibly combining several into one, and selects one regarded as most appropriate.
- It applies the scheme to the formula at hand, uses simple propositional calculus to normalize the result, and puts each of the new formulas back into the pool.
- Propositional calculus normalization may make the instantiation of the induction scheme look different than the scheme itself. For example, instead of $(q \wedge (\alpha' \Rightarrow \beta')) \Rightarrow (a \Rightarrow \beta)$, propositional normalization produces two formulas: $(q \wedge \neg\alpha' \wedge \alpha) \Rightarrow \beta$ and $(q \wedge \beta' \wedge \alpha) \Rightarrow \beta$.
- It is possible to prove an induction rule (see induction) so that a term suggests other inductions.
- You can override its choice of induction by supplying an induction hint.

Simplification Overview

- Simplification is the heart of the theorem prover. It:
  - applies propositional calculus, equality, and linear arithmetic decision procedures,
  - uses type information and forward chaining rules to construct a “context” describing the assumptions of each subterm,
  - rewrites each subterm in the appropriate context, using definitions, conditional rewrite rules, and metafunctions,
  - uses propositional calculus normalization to convert the resulting formula to an equivalent set of formulas, reduces the set under subsumption, and deposits the surviving formulas back in the pool.
- The simplifier is not guaranteed to produce formulas that are stable under simplification; repeated trips through the simplifier, via insertion and extraction from the pool, are used to reach the final stable form (if any).

Destructor Elimination

- Elim rule example: suppose a formula mentions `(CAR A)` and `(CDR A)`. If `A` is a cons, we could replace `A` by `(CONS A1 A2)`, for new variables `A1` and `A2`, allowing us to replace `(CAR A)` and `(CDR A)` with `A1` and `A2`.
- `CAR-CDR-ELIM` axiom: `(=> (consp x) (== (cons (car x) (cdr x)) x))`
- This axiom is an example of a more general form:
  - `(=> (hyp x) (== (constructor (dest1 x) . . . (destn x)) x))`
  - Such theorems can be stored as “destructor elimination” or elim rules.
  - The `(desti x)` are the destructor terms.
- Applies when a formula contains an instance of `(desti x)` and `x` is bound to a variable, say *a*.
- It “splits” the formula into two, according to whether `(hyp` *a*`)` is true; when true, it replaces all of the *a*’s in the formula (except those inside `desti` applications) by (constructor `(dest1` *a*`) . . . (destn` *a*`)`).
- Replaces all the `(desti` *a*`)` terms with distinct new variable symbols, *a*1, … , *a*n.

Use of Equivalences

- If the formula contains the hypothesis `(== lhs rhs)` and elsewhere in the formula there is an occurrence of `lhs`, then `rhs` is substituted for `lhs` in every such occurrence based on heuristics.
- ACL2 supports a more general form of substitution involving equivalence relations. The use of equalities is generalized to the use of any equivalence relation.

```
(=> (^ (== (rev (rev a2)) a2)
       (tlp a2))
    (== (rev (app (rev a2) (list a1)))
        (cons a1 a2)))
```

$\Rightarrow$

```
(=> (tlp a2)
    (== (rev (app (rev a2) (list a1)))
        (cons a1 (rev (rev a2)))))
```

Generalization

- Find a subterm that appears in both the hypothesis and the conclusion, in two different hypotheses, or on opposite sides of an equivalence
- Replace that subterm by a new variable symbol
- If type information (see type-prescription) or generalization rules (see generalize) can be used to restrict the type of the new variable, then it is so restricted. The generalized formula is then added to the pool.

```
(=> (tlp a2)
    (== (rev (app (rev a2) (list a1)))
        (cons a1 (rev (rev a2)))))
```

$\Longrightarrow$

```
(=> (tlp a2)
    (== (rev (app rv (list a1)))
        (cons a1 (rev rv))))
```

Elimination of Irrelevance

- Eliminate irrelevant hypotheses, by partitioning them into cliques according to the variables they mention.
- If there are isolated cliques of hypotheses, then either the formula is a theorem because those hypotheses are collectively false, or else they are irrelevant.
- Use type information to show that a clique is not false.

```
(=> (tlp a2)
    (== (rev (app rv (list a1)))
        (cons a1 (rev rv))))
```

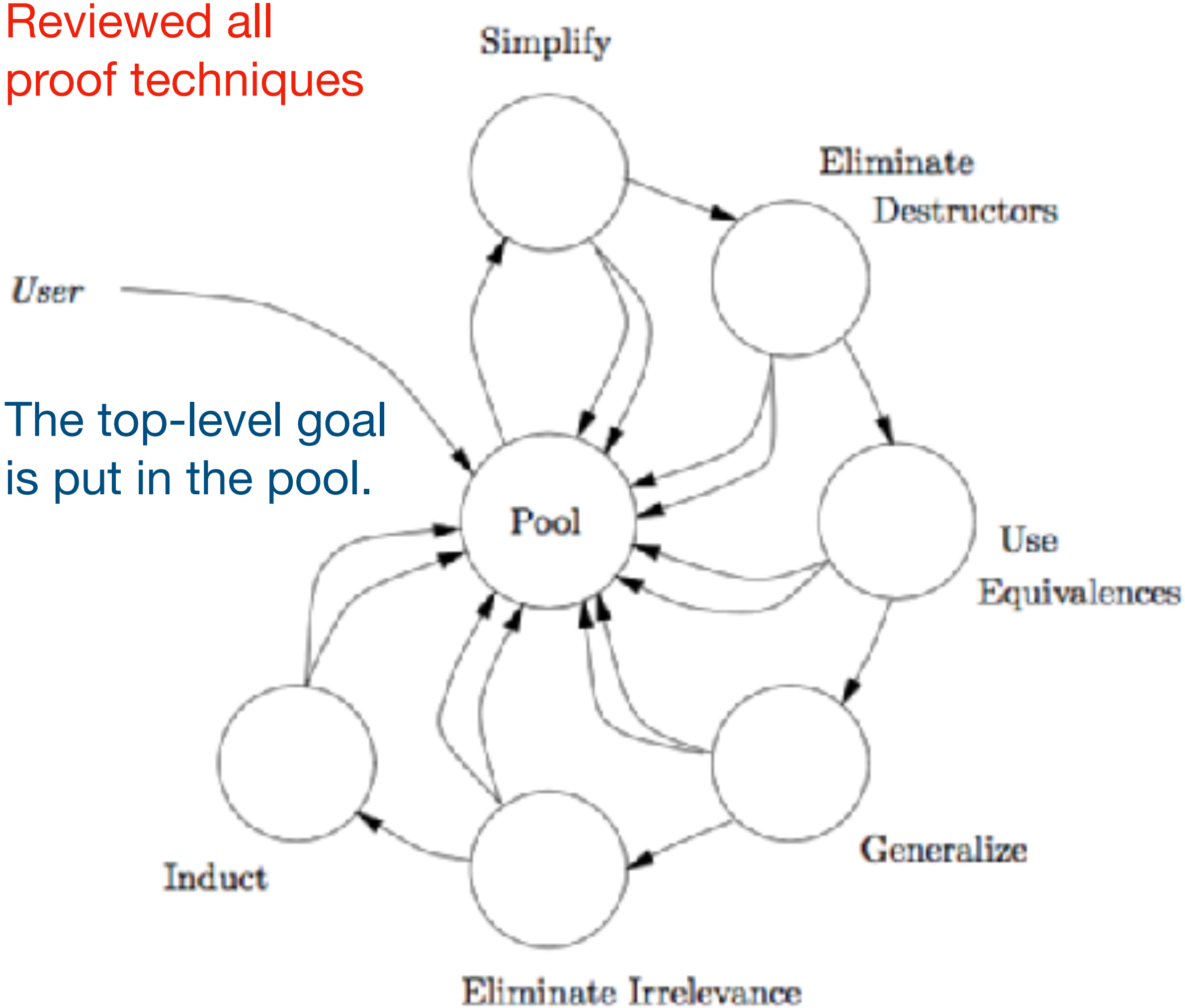
$\Rightarrow$

```
(== (rev (app rv (list a1)))
    (cons a1 (rev rv)))
```

Organization of ACL2

Reviewed all
proof techniques

The top-level goal
is put in the pool.



When a formula is drawn out, it is passed to proof techniques until one applies.

The draw is orchestrated that we do not try to prove a subgoal by induction until we have processed every subgoal produced by the last induction.

Slides by Pete Manolios for CS4820

Questions?