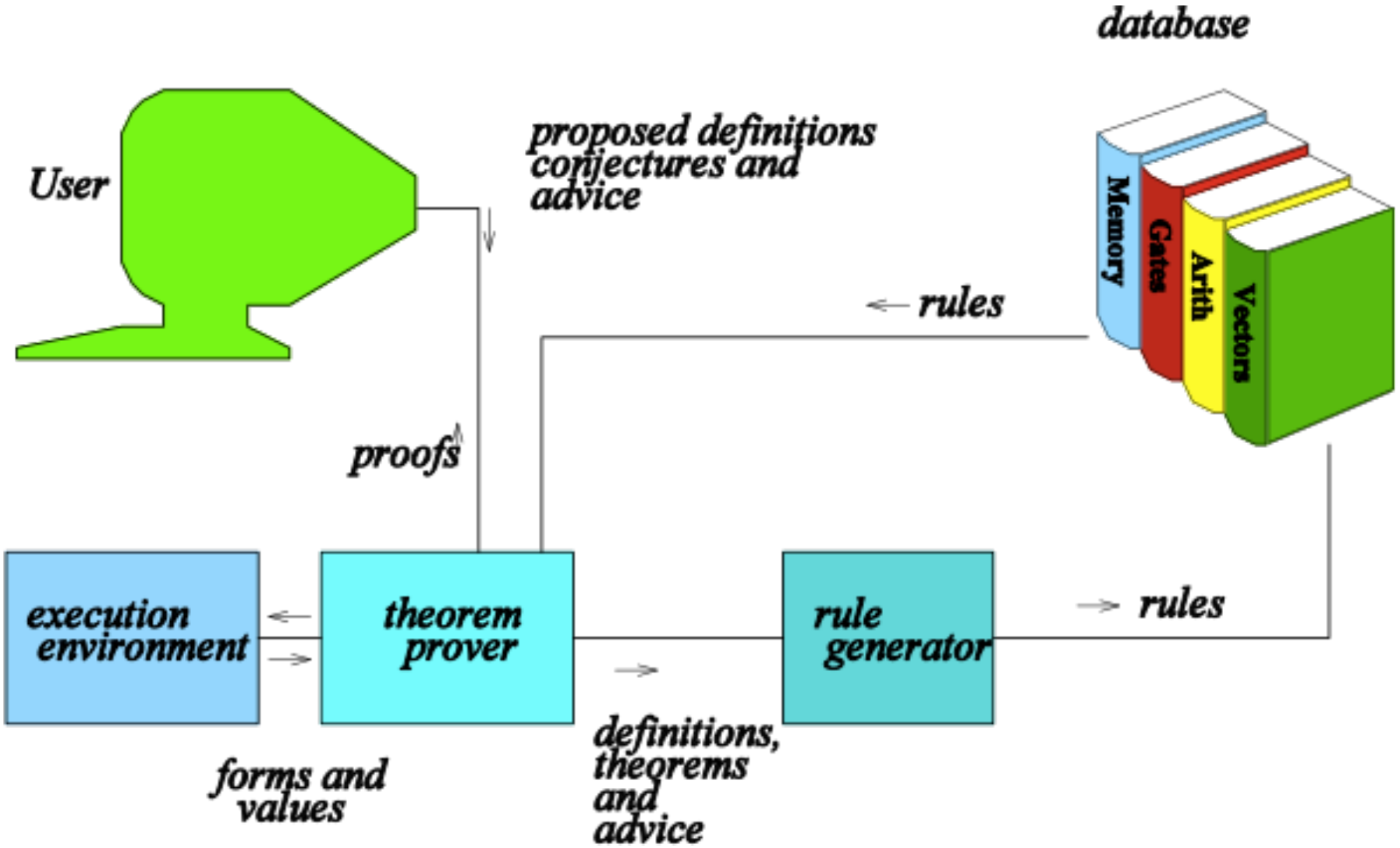
Lecture 5

Pete Manolios
Northeastern

ACL2 is . . .

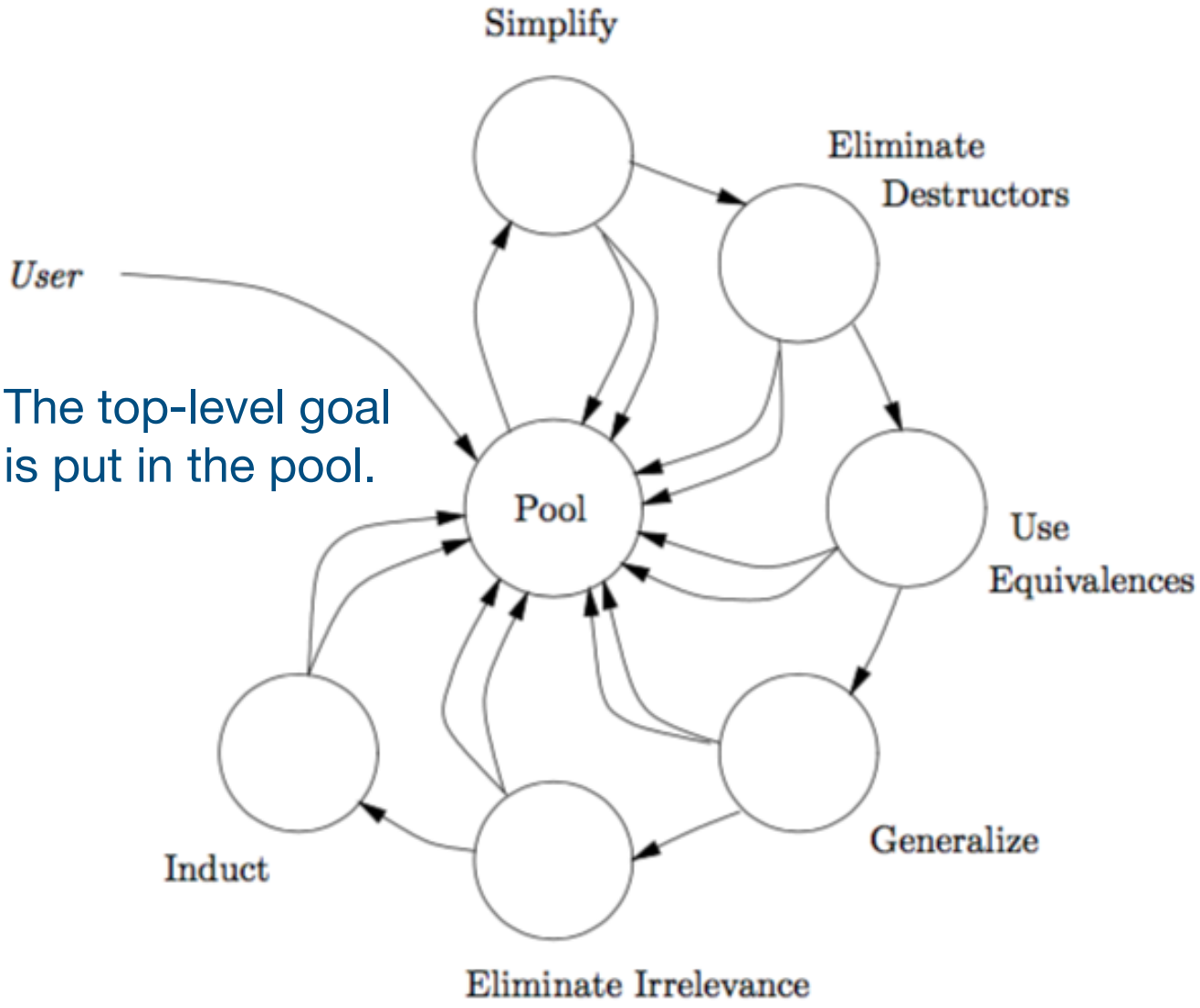
- **A programming language:**
  - Applicative, functional subset of Lisp
  - Compilable and executable
  - Untyped, first-order
- **A mathematical logic:**
  - First-order predicate calculus
  - With equality, induction, recursive definitions
  - Ordinals up to ε0 (termination & induction)
- ***A mechanical theorem prover:***
  - *Integrated system of ad hoc proof techniques*
  - *Heavy use of term rewriting*
  - *Largely written in ACL2*

ACL2 System Architecture



Slides by Pete Manolios for CS4820

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 $(q \wedge (\alpha' \rightarrow \beta')) \rightarrow (\alpha \rightarrow \beta)$, 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: (implies (consp x) (equal (cons (car x) (cdr x)) x))
- This axiom is an example of a more general form:
  - (implies (hyp x) (equal (constructor ($dest_1$ x) . . . ($dest_n$ 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 ($dest_i$ *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 $dest_i$ applications) by (constructor ($dest_1$ *a*) . . . ($dest_n$ *a*)).
- Replaces all the ($dest_i$ *a*) terms with distinct new variable symbols, a_1, … , a_n.

Use of Equivalences

- If the formula contains the hypothesis (equal 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.

```
(implies
 (and (equal (rev (rev a2)) a2)
      (true-listp a2))
 (equal (rev (app (rev a2) (list a1)))
        (cons a1 a2)))
```

```
(implies
 (true-listp a2)
 (equal (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.

```
(implies
 (true-listp a2)
 (equal (rev (app (rev a2) (list a1)))
        (cons a1 (rev (rev a2)))))
```

```
(implies
 (true-listp a2)
 (equal (rev (app rv (list a1)))
        (cons a1 (rev rv))))
```

Elimination of Irrelevance

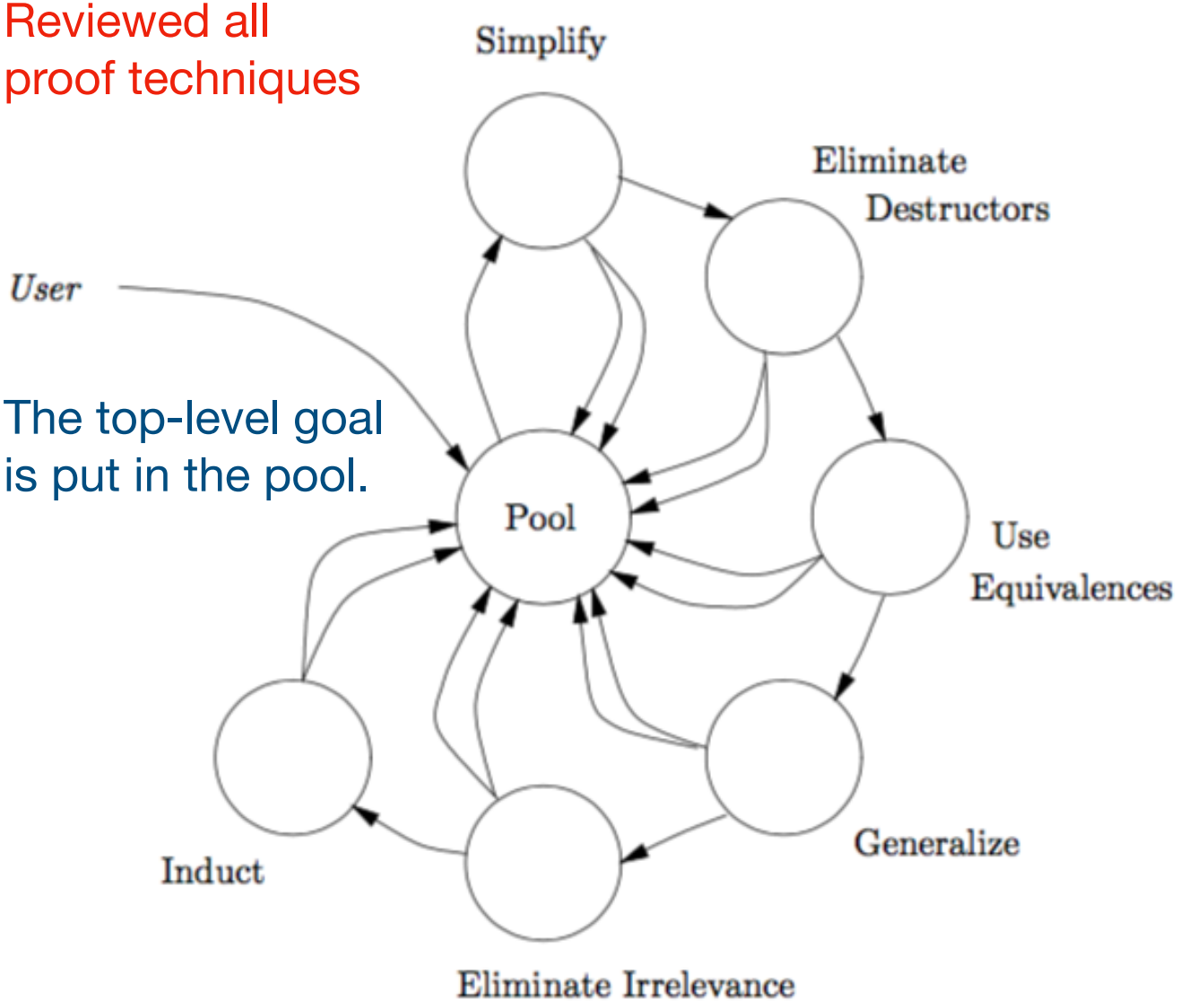
- Eliminate irrelevant hypotheses, by partitioning them into cliques according to the variables they mention.
- If there are isolated cliques of hypotheses, then either the formula is a theorem because those hypotheses are collectively false, or else they are irrelevant.
- Use type information to show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

```
(implies (true-listp a2)
         (equal (rev (app rv (list a1)))
                (cons a1 (rev rv))))
```

```
(equal (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.

Rev-rev demo & log

Slides by Pete Manolios for CS4820

Simplification in Detail

- 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,
  - use propositional calculus normalization to convert the resulting formula to an equivalent set of formulas, reduce the set under subsumption, and deposit the surviving formulas back in the pool.
- Assume the formula to which the simplifier is applied is of the form (implies (and p_1 . . . p_n) q). The pi are the hypotheses and q is the conclusion.
- First we discuss equivalence relations and congruence rules, which are fundamental to several aspects of the simplifier.
- Then we discuss each of the four steps in the order in which they occur.

Congruence-Based Reasoning

- General form of substitution of equals for equals based on the ideas of user-defined equivalence relations and congruence rules.
- Consider: (implies (set-equal x y) (iff (member e x) (member e y)))
- This congruence rule allows the substitution of set-equals for set- equals, in the second argument of member expressions, while preserving iff.
- Use defequiv to identify equivalence relations.
- We say that a term of a formula is equiv-hittable if congruence rules establish it can be replaced by any equivalent term without changing the propositional value of the formula.
- Congruence rules: (implies (equiv1 x y) (equiv2 (f ... x ...) (f ... y ...))) where equiv1 and equiv2 are known equivalence relations.
- Use defcong to prove congruence rules.
- The rule allows equiv1 substitution into f while preserving equiv2. ACL2 justifies deep substitutions by chaining together congruence rules, starting from a rule that preserves iff (propositional equivalence).

Decision Procedures

- When a formula is given to the simplifier three decision procedures are applied.
- Propositional Calculus based on the normalization of if expressions
  - Propositional connectives are expanded in terms of if
  - the if terms are distributed, so (f (if a b c)) becomes (if a (f b) (f c)) and (if (if a b c) x y) becomes (if a (if b x y) (if c x y))
  - the resulting tree is explored to determine whether every reachable tip is non-nil.
- Congruence Closure: use the context to compute equivalence classes, choose a representative per equivalence class, and substitute that representative for all members of the class. Repeat until fixpoint is reached.
- Rational linear arithmetic: linear data base contains all inequalities (<,<=, >=,>,=) relevant to conjecture, where function applications other than sums, differences, and products with constants are treated as variables.
  - *Linear rules* are theorems that concludes with an inequality. If an instance of one of the terms in the inequality arises in the linear data base, the rule is instantiated