ACL2

A Computational **L**ogic
for
Applicative **C**ommon **L**ISP

History

- 1971-1992: Boyer-Moore system/Nqthm
  - *A Computational Logic,* Boyer, Moore (1979)
- 1994-*present:* ACL2
  - First publication in 1994
    - *Design Goals of ACL2,* Kaufmann, Moore
  - Latest publication in 2010
    - Buchberger's algorithm in LISP
  - Current version: 3.6.1 (september 2009)

Success stories

- Verification of AMD floating point units
  - Intel Pentium: FDIV bug!
  - AMD K5: microcode formalized in ACL2
    - Moore, Kaufmann, Lynch (1995)
  - AMD Athlon: RTL translated into ACL2
    - Russinoff, Flatau (1998)

Key points

- ACL2 language derives from LISP
  - No side-effects
  - All functions must be total
- Quantifier-free first order logic (mostly!)
  - Implicit universal quantification
  - Well-founded strong induction
- High degree of automation
  - ACL2 does tedious proofs for you

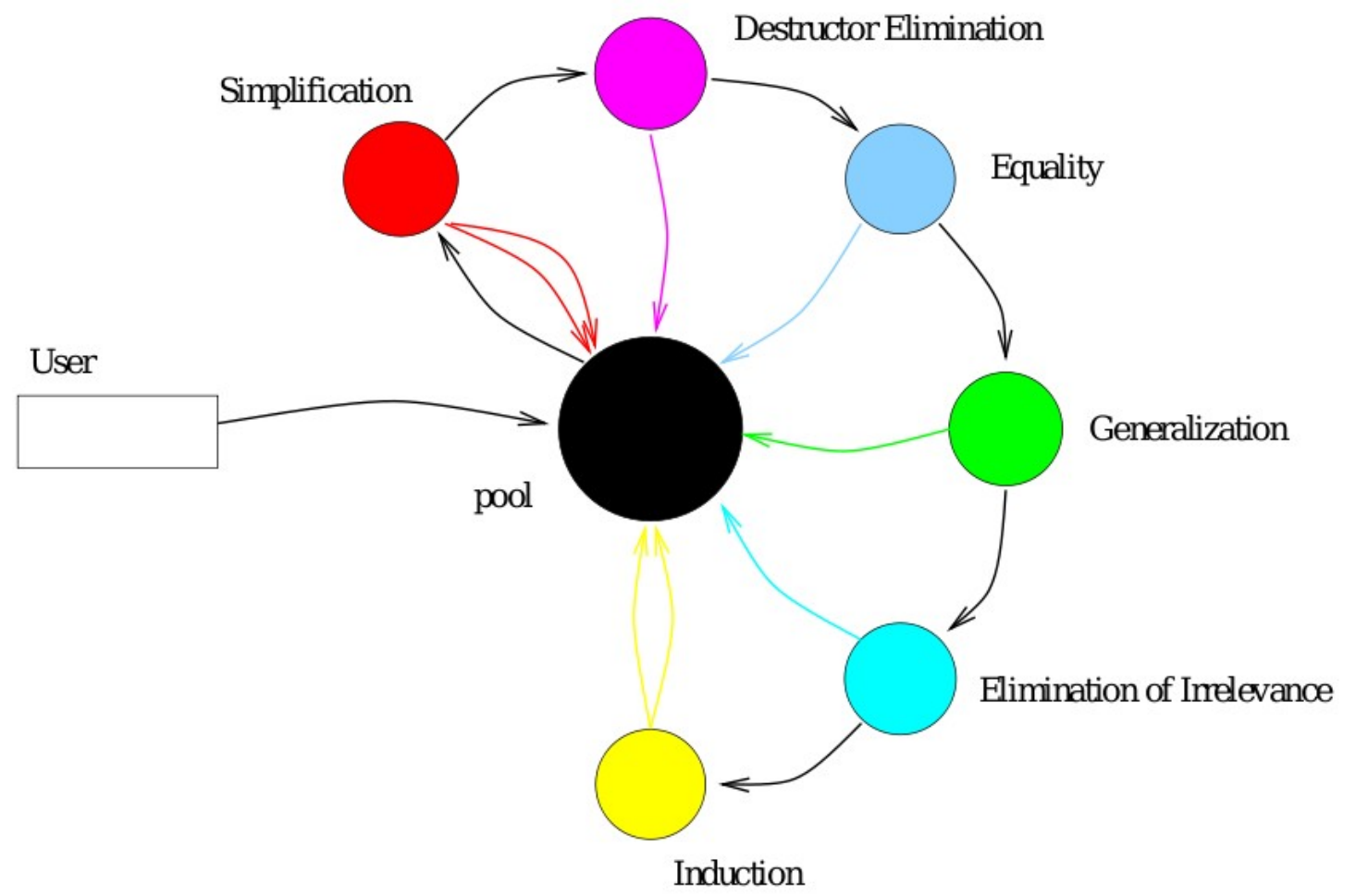
”Industrial strength”

- Advantages
  - Abundant, accessible documentation
  - Interface provides a lot of feedback
  - Well-supported (*ACL2 Sedan)*
- Drawbacks:
  - Steep initial learning curve
  - No 'kernel': soundness bugs do occur
  - Logic restrictive for mathematical proofs

The Method

1) Think about the proof of your theorem
2) Let ACL2 attempt to prove it
3) Success: go to next theorem, else:
4) Examine the failed proof attempt and:
   - Fix your theorem!
   - Discover lemmas ACL2 needs
   - Provide hints to the prover

The Waterfall



(Source: *ACL2 Theorems about Commercial Microprocessors*)

LISP primer

CAML

```
f (x+2)*3
```

LISP

LISP primer

CAML

```
f (x+2)*3
```

LISP

```
(* (f (+ x 2)) 3)
```

LISP primer

CAML

```
f (x+2)*3
```

```
[1; 2; 3; 4]
```

LISP

```
(* (f (+ x 2)) 3)
```

```
'(1 2 3 4)
```

LISP primer

CAML	*LISP*
`f (x+2)*3`	`(* (f (+ x 2)) 3)`
`[1; 2; 3; 4]`	`'(1 2 3 4)`
`x::y`	`(cons x y)`
`hd x`	`(car x)`
`tl x`	`(cdr x)`

LISP primer

CAML

```
f (x+2)*3
[1; 2; 3; 4]
x::y
hd x
tl x
let square x = x*x
```

LISP

```
(* (f (+ x 2)) 3)
'(1 2 3 4)
(cons x y)
(car x)
(cdr x)
(defun square (x)
   (* x x))
```

Examples

```
(defun fac (x)
   (if (zerop x) 1
       (* x (fac (1- x)))))

(defthm fac-grows
   (implies (< 2 x) (< x (fac x))))
```

$$\forall x.\, 2 < x \rightarrow x < x!$$

```
(defun append (x y)
   (if (endp x) y
       (cons (car x) (append (cdr x) y))))

(defun reverse (x)
   (if (endp x) '()
       (append (reverse (cdr x)) (list (car x)))))
```