Building Interactive Libraries of Formal Algorithmic Knowledge

Stuart F. Allen – Cornell University

James L. Caldwell – University of Wyoming

Robert L. Constable – Cornell University

Jason J. Hickey – Cal Tech

Arlington, Virginia – July 10, 2001

project url: www.cs.cornell.edu/Info/Projects/NuPRL/nuprl.html

Introduction

"The world is engaged in a grand scientific and technological enterprise to build a global information resource. It will include vast digital collections of scientific information. Creating this global resource will be one of the great achievements of information technology."

-- Cornell proposal to DoD responding to ONR

Introduction

- The organization of scientific information on a large scale is an idea whose time has come
- Paul Ginsparg's arXiv is a glimpse of the future as is the Nuprl library of formal computational mathematics
- I will talk about ideas that will relate these two developments
- We foresee advances in this field which will rank as landmark achievements of information technology

Outline

1. Introduction
2. Logical infrastructure
3. Sharing Knowledge
4. Conclusion

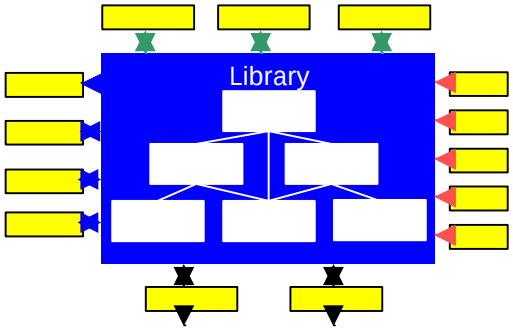
Outline

1. **Introduction**
   - Goals
   - Value
   - Problems
   - Approach
2. Logical infrastructure
3. Sharing Knowledge
4. Conclusion

Project

Our project has two complementary aspects:

1. Building a logically sound information management service
2. Providing formal content for a digital library of computational mathematics



Value

The logical infrastructure will add considerable value to a digital library of mathematics, making numerous applications possible that are important to the Department of Defense.

For instance:

1. the library can store important reference versions of algorithms along with demonstrations that they have certain properties
2. formally documented computer systems can be stored as “books” in the library (see our Darpa projects for examples, e.g. Ensemble distributed group communications system; new work on event notification services)

Value

3. educational materials for mathematics and computer science will be part of the library

4. the library will accelerate research in all technical fields, especially in software

5. the library will be a nucleus for numerous information services, such as searching, indexing, summarizing and translating

Value Summary

Major contribution to our capability to learn, understand, discover and teach.

Major opportunity to focus our tools on a fundamental human activity – exploring and understanding

Also, a way to protect, preserve and extend one of humankind's most positive legacies – organized knowledge

Background of the project team

- Many years spent building and using tools to improve hardware and software reliability
- This work has generated:
  - large database of hyperlinked mathematical facts (about 7K definitions and theorems)
  - two systems used in the work:
    -- Nuprl 5
    -- MetaPRL (with JProver)
  - initial explorations of a combined database from Nuprl, MetaPRL, HOL; we call this a Logical Library

Outline

1. Introduction
2. **Logical infrastructure**
   - Need for expressiveness of inferences
   - Problems with expressiveness
   - Possible solutions
3. Sharing Knowledge
4. Conclusion

Logical issues

We know how to usefully formalize propositions.
The challenge is to usefully formalize proofs.

We focus on natural expressiveness of inferences

- key to intelligibility
- key to readability
- basis of a proof technology

Logical issues

We focus on generality of language and inference

- type theory is known to be general
- covers wide range of applications
- language expressiveness helps proof expressiveness

Pigeon Hole Principle Theorem

For example, if 7 pigeons are put in a smaller number of holes, then at least 2 pigeons will share a hole.

Generally, if we map m objects into n slots, and m is bigger than n, then two objects share a slot.

Define $\mathbb{N}m = \{0, 1, 2, \ldots, m-1\}$.

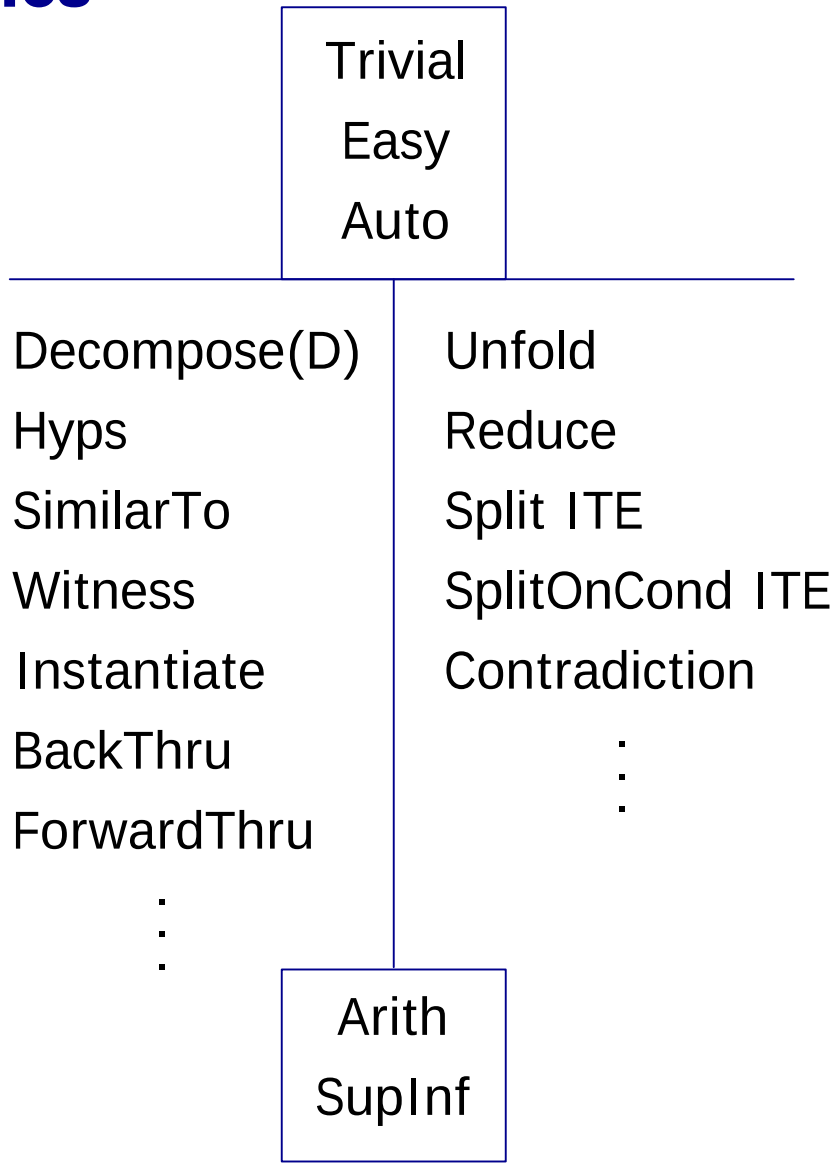
Theorem If f maps $\mathbb{N}m$ to $\mathbb{N}n$ where $m>n$, then there are i, j in $\mathbb{N}m$ such that $i<j$ and $f(i)=f(j)$.

Responding to complexity

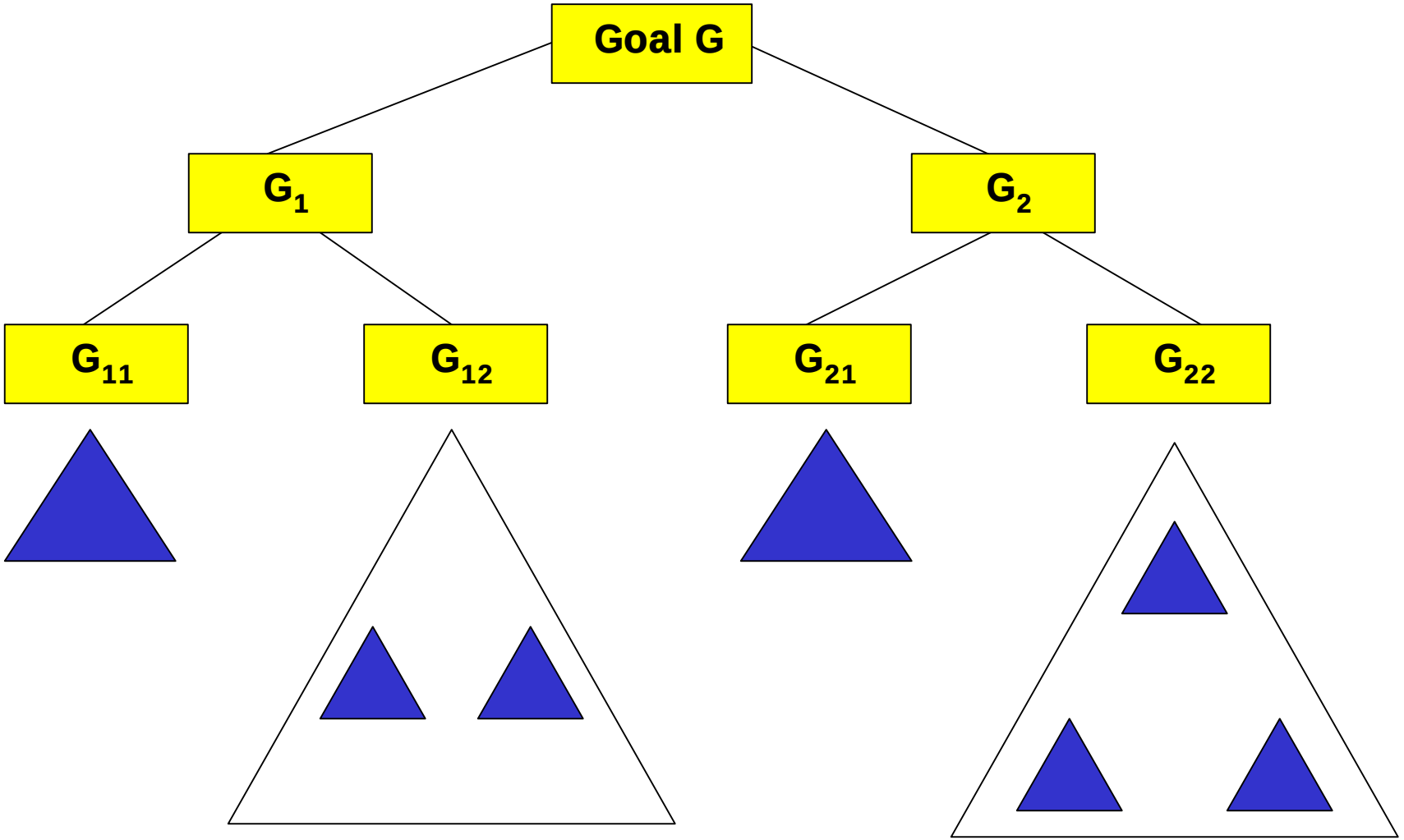
The computer science response to the complexity caused by expressiveness and generality has been the use of programs to manage the complexity “behind the scenes.” In the case of proofs, this code is called tactics.

A tactic is a program that builds part of a proof.

List of tactics



Tactics



Complications

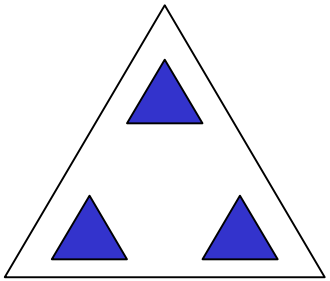
But these features complicate some natural operations on objects (def, thms, proofs), such as tracking dependency

Let's see why

Expressiveness and generality lead to a great deal of checking; 10 steps blow up to approx 200 (discouraged people at first)

Expressive Inferences – the state problem

tactic-tree proof

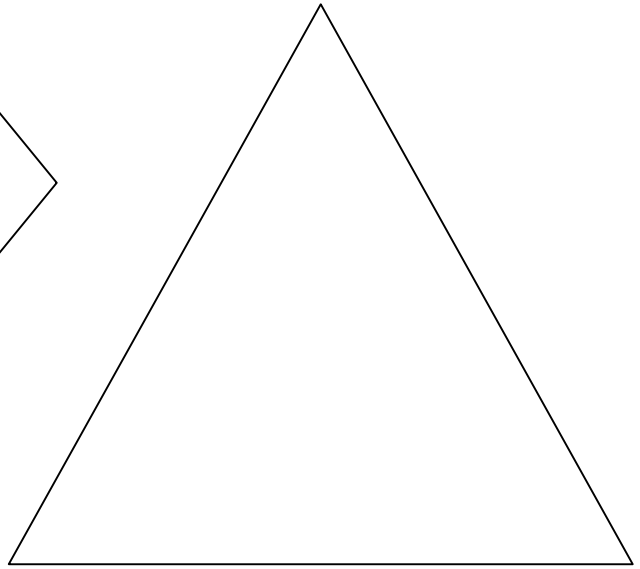


expand

Library

ML-state

primitive proofs



Check primitive proof independently – U. Wyoming using ACL2

Logical complications from tactics

The more tactics look like standard programs, the more work they can do, but the more logically intractable they are, e.g. the more information they hold in state.

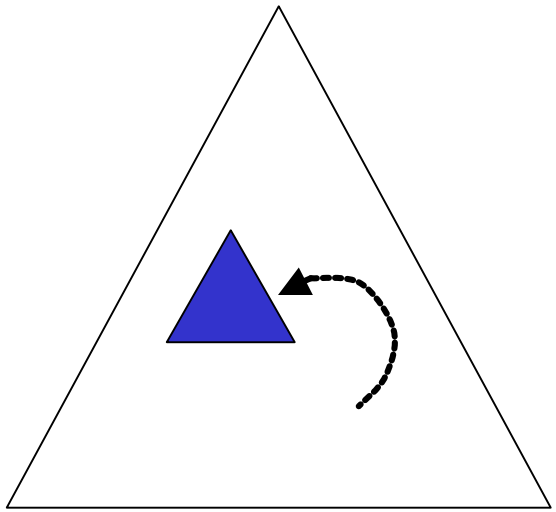
Source of state

- Library
  - -- search for relevant theorem (easy-tactic)
- Global M-L variables
  - -- “Trivial”
  - -- “type check”
  - -- “auto”
- Nominal caches
  - -- wff-proofs
  - -- equality proofs

The state of a proof

Library

- rule
- def
- fact
- lemma
- tactic
- theorem



ML-state

global-var1
global-var2
global-var3
.
.
.

Logical library must manage proof state

Possible solutions

- Create stable tactics
  - -- controlled library access
  - -- read-only global variables
  - -- real caches
- Transform tactics to stable ones
- Check primitive proofs

Pigeon Hole Principle – Informal Proof

Recall that $\mathbb{N}m = \{0, 1, 2, \ldots, m-1\}$.

Theorem If f maps $\mathbb{N}m$ to $\mathbb{N}n$ where m>n, then there are i, j in $\mathbb{N}m$ such that i<j and f(i)=f(j).

Proof:

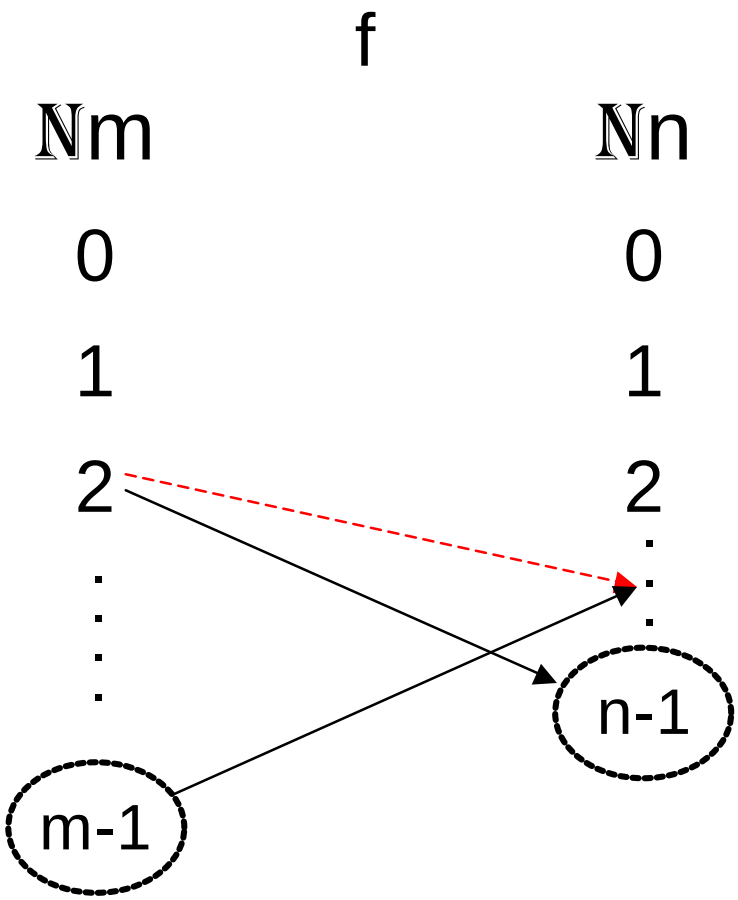
In the case where n=1, $\mathbb{N}n$ has exactly one element, 0;
so for any m>1 and any f, f(x)=f(y)=0.

We can reduce n by 1 until we reach this base case, preserving the existence of a mapping and recovering the original f from any of the others.

$$\mathbb{N}_n \quad \mathbb{N}_{n-1} \quad \cdots \quad \mathbb{N}_1$$
$$f \quad f_{n-1} \quad f_1$$

(continued)

Plan, argue by induction on n. Remove the largest element of $\mathbb{N}n$ and build a function f' from $\mathbb{N}m-1$ to $\mathbb{N}n-1n$ in such a way that if f'(x) =f'(y), then f(x') = f(y').



$f'(x_0) = f(m-1)$

(continued)

Assume for induction that for f mapping $\mathbb{N}m$ to $\mathbb{N}n-1$ for $m>n-1$, there are x, y in $\mathbb{N}m$, $x<y$ and $f(x)=f(y)$.

To build f', see if any x maps to n-1.

($\exists x:\mathbb{N}m.\ f(x)=n-1$ is decidable.)

If not, then f actually maps $\mathbb{N}m$ to $\mathbb{N}n-1$ and by induction hypothesis there are x, y in $\mathbb{N}m$ such that $f(x)=f(y)$, $x<y$.

(continued)

Let x_0 be the least element of $\mathbb{N}m$ such that $f(x)=n-1$.

See if any $y>x_0$ satisfies $f(y)=n-1$.

($\exists y:\mathbb{N}m.\ y>x_0$ & $f(y)=n-1$ is decidable.)

If there is a y, then take $x'=x_0$, $y'=y$ and note that $f(x')=f(y')$, $x'<y'$.

If there is no such y, then let f' be the same as f except at x_0 define $f'(x_0)=f(m)$.

(continued)

Notice that f' maps $\mathbb{N}m-1$ to $\mathbb{N}n-1$ since all values of f' are less than n-1. Thus the induction hypothesis applies and we have x, y in $\mathbb{N}m-1$, $x<y$ and $f'(x)=f'(y)$.

If x or y is x_0, without loss of generality (wlg) say x is x_0, then take $x'=x$, $y'=m$, and note $f(x')=f(y')$.

If neither x or y is x_0, then clearly $f(x')=f(y')$, since f' and f agree except on x_0.

Qed.

(continued)

Another approach to Pigeon Hole Principle

The previous proof leads to exponential runtime if implemented as presented. There are more efficient proofs.

Given f mapping $\mathbb{N}$m to $\mathbb{N}$n, we can check for each element of {0, 1, ... , m-1} whether it maps to the same element as a previous value. The code could be

```
do i=0 to m-1
   do j=i+1 to m-1
      if f(j)=f(i) then return <i,j>
   end
end
```

(continued)

(Another Approach continued)

```
do i=0 to m-1
   do j=i+1 to m-1
      if f(j)=f(i) then return <i,j>
   end
end
```

If the outer loop completes, then f is an injection. We know this key fact.

Theorem If f is an Injection from $\mathbb{N}$m to $\mathbb{N}$n, then m ≤ n.

Since we assume m>n in Pigeon Hole, f cannot be an injection, so a pair <i,j> must be returned.

Some Definitions and Lemmas used in the Proof of pigeon_hole

```
Def Inj(A; B; f) == ∀a1,a2:A. f(a1) = f(a2) ∈ B ⇒ a1 = a2

(finite_inj_counter_example)
Thm* ∀m:ℕ, f:(ℕm→ℤ).
      ¬Inj(ℕm; ℤ; f) ⇒ (∃x:ℕm, y:ℕx. f(x) = f(y))

(inj_imp_le)
Thm* (∃f:(ℕm→ℕk). Inj(ℕm; ℕk; f)) ⇒ m≤k
```

Pigeon Hole

```
├ ∀m,k:ℕ, f:(ℕm→ℕk). k<m ⇒(∃x,y:ℕm. x ≠ y & f(x) = f(y)) by Auto
|
1. m : ℕ
2. k : ℕ
3. f : ℕm→ℕk
4. k<m
├ ∃x,y: ℕm. x ≠ y & f(x) = f(y)
| by Inst: →finite_inj_counter_example on Tms:[m ; f] ....
|
|\
| .....antecedent.....
| ├ ¬Inj(ℕm; ℤ; f) by ¬m≤k Asserted ...w
| |
| 5. ¬m≤k
| ├ ¬Inj(ℕm; ℤ; f) by SimilarTo -1 ....
| |
| 5. Inj(ℕm; ℤ; f)
| ├ m≤k by BackThru: →inj_imp_le ....
| |
| ├ ∃f:(ℕm→ℕk). Inj(ℕm; ℕk; f) by Witness: f ....
| |
| ├ Inj(ℕm; ℕk; f) by SimilarTo: 5 ....
|
.\
. 5. ∃x:ℕm, y:ℕx. f(x) = f(y)
. ├ ∃x,y:ℕm. x ≠ y & f(x) = f(y) by SimilarTo: -1 ....
```

Nuprl Section: DiscreteMath – Discrete Math Lessons

Counting is finding a function of a certain kind.

When we count a class of objects, we generate an enumeration of them, which we may represent by a 1-1 CORRESPONDENCE from a standard class having that many objects to the class being counted. Our standard class of n objects, for $n \in \mathbb{N}$, will be $\mathbb{N}n$, which is the class $\{k: \mathbf{Z} \mid 0 \leq k < n\}$ of natural numbers less than n.

So, a class A has n members just when

$$\exists f:(\mathbb{N}n \to A).\ \mathrm{Bij}(\mathbb{N}n; A; f)$$

Which may also be expressed as

$$(\mathbb{N}n \sim A).$$

(continued)

(continued)

We may ask whether counting in different ways, i.e., coming up with different orders, will always result in the same number, as we assume. Of course, we know this is so, but there are different degrees of knowing. It is not necessary to simply accept this as an axiom; there is enough structure to the problem to make a non-trivial proof.

Thm* $(A \sim \mathbb{N}m) \Rightarrow (A \sim \mathbb{N}k) \Rightarrow m = k.$

This theorem is closely related to what is sometimes called the “pigeon hole principle,” which states the mathematical content of the fact that if you put some number objects into fewer pigeon holes, then there must be at least two objects going into the same pigeon hole.

Thm* $\forall m, k{:}\mathbb{N}, f{:}(\mathbb{N}m \to \mathbb{N}k).\ k<m \Rightarrow (\exists x,y{:}\ \mathbb{N}m.\ x \neq y \ \&\ f(x) = f(y))$

If you examine the proofs of these theorems, you will notice that they both cite the key lemma

Thm* $(\exists\, f{:}(\mathbb{N}m \to \mathbb{N}k).\ \mathrm{Inj}(\mathbb{N}m; \mathbb{N}k; f)) \Rightarrow m \leq k.$ *(inj_imp_le)*

Outline

1. Introduction
2. Logical infrastructure
3. **Sharing Knowledge**
   - Need to share
   - Problems with sharing
   - Possible solutions
4. Conclusion

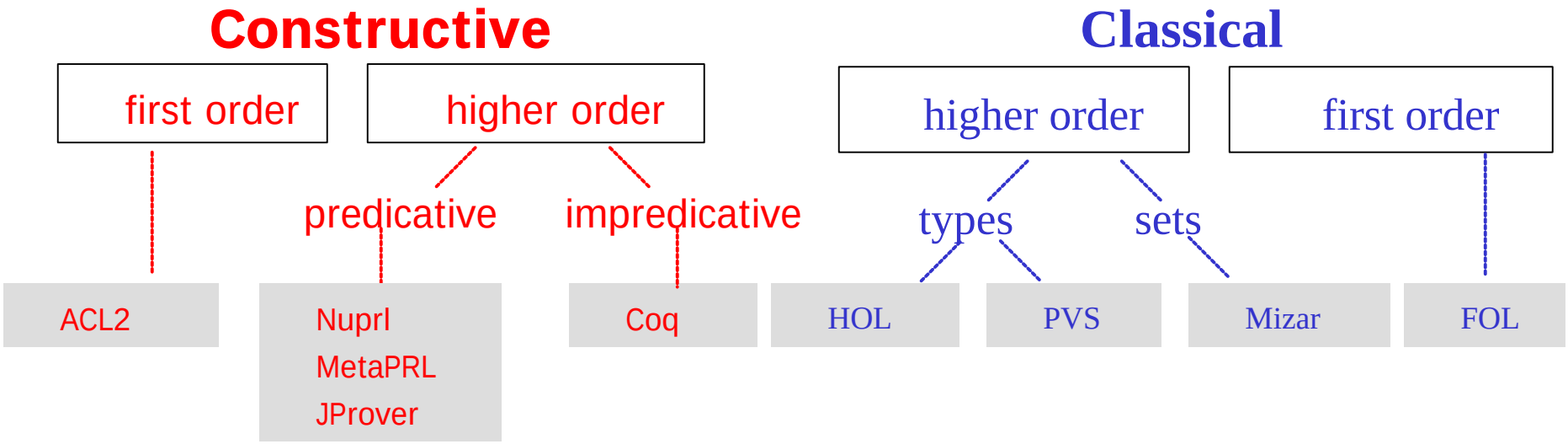
Sharing mathematical knowledge

- Building a large resource requires global cooperation
- Cooperation entails sharing among provers

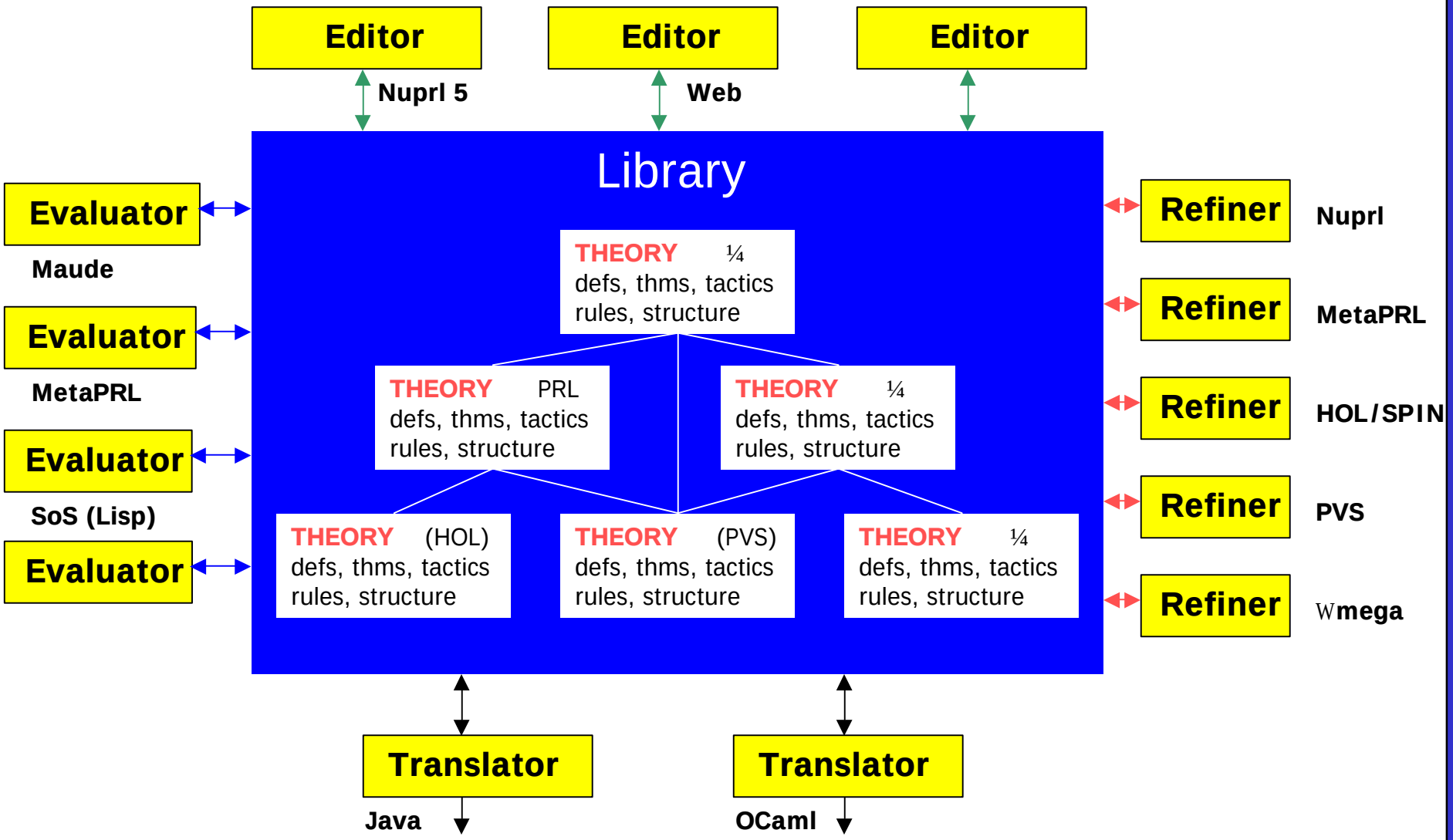
The problem of sharing knowledge

There will always be several logics, several provers.

For example, one standard classification:



The Nuprl LPE Architecture



Possible Solutions – Proof Sentinels

To limit proofs to a given set of rules, axioms, and proof resources specific to a given theory, we will use resource expressions and proof sentinels to monitor compliance with resource limitations. We have experience with these mechanisms in MetaPRL.

Typical examples are:

core Nuprl	(compatible with Alf)
class theory	(Nuprl with $\sqsubseteq$, $\cap$, $\cup$)
domain theory	(Nuprl with bar types)
classical Nuprl	
const. set theory	
HOL	
FOL	

Outline

1. Introduction
2. Logical infrastructure
3. Sharing Knowledge
4. **Conclusion**
   - Other issues
   - Work plans
   - Expectations

Formal content

We focus on formal computational mathematics

1. needed in software reliability
2. correct to highest standards known
3. basis for information services not available with text, hence opportunity for experiments to determine what to extend to text
4. a tool of great precision to analyze information content

Subcontracts

- Cal Tech – Jason Hickey

  Jason is contributing to the MetaPRL system. The Nuprl/MetaPRL link is our best mechanism for sharing knowledge among theories and systems.

  We are in weekly contact about MetaPRL.

- University of Wyoming – James Caldwell

  Jim and his colleagues are contributing new formal content, e.g. propositional logic, rationals and reals. They are also building a primitive proof checker in ACL2.

Library operations

- *Building a new (computational) theory – basic operations*
  - name and create axioms, rules, definitions, theorems, etc.
  - prove theorems (possibly using multiple provers)
  - provide logical accounting for each inference step
  - explicitly store justifications of inference steps
  - create new tactics
  - add new decision procedures
  - compile abstract algorithms to standard languages
  - link formal objects to text
  - search for objects based on form, content and metadata
  - present material in various formats (TeX, HTML)
  - browse a theory
  - manage and maintain multiple representations of objects

Library operations

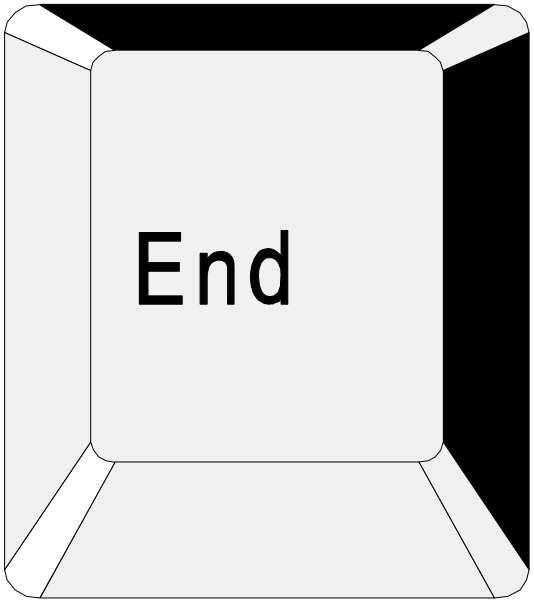
- *Archiving a theory – more advanced operations*
  - store objects in a form that is *stable* with respect to future extensions
  - compute logical dependencies in library
  - classify results with respect to standard theories
  - link to records of development and authorship (such as arXiv)
  - provide readings of the results
  - certify critical objects by authoritative human review

Library operations

- *Modifying a theory – advance version control operations*
  - change justifications of inference steps and automatically recheck
  - change definitions and theorems and propagate significant changes to objects
  - replay proofs in new environment
  - modify proofs and automatically propagate change
  - re-root a theorem or a subtheory into a different theory
  - add and change text links

Library operations

- *Operate on theories – advanced large-scale operations*
  - combine theories (union, intersection)
  - find all dependencies
  - prune theories around main theorems
  - reflect a theory
  - generalize a theory
  - specialize a theory
  - translate one theory into another
  - mine a theory for new relationships, connections and proof methods


End