Integrating Math Units and Proof Checking for Specification and Verification

Hampton Smith
Kim Roche
Murali Sitaraman
Clemson University

Joan Krone
Denison University

William F. Ogden
Ohio State University

SAVCBS Workshop 2008
SIGSOFT 2008 / FSE 16
November 9th, 2008

Overview

- RESOLVE Verification System
- Role of Proof Checker in Verification System

- Requirements of a Proof Checker in such a system

Overview

- RESOLVE Verification System
- Role of Proof Checker in Verification System
  - Issues
  - Solutions
- Requirements of a Proof Checker in such a system
  - Issues
  - Solutions

RESOLVE Verification System

RESOLVE

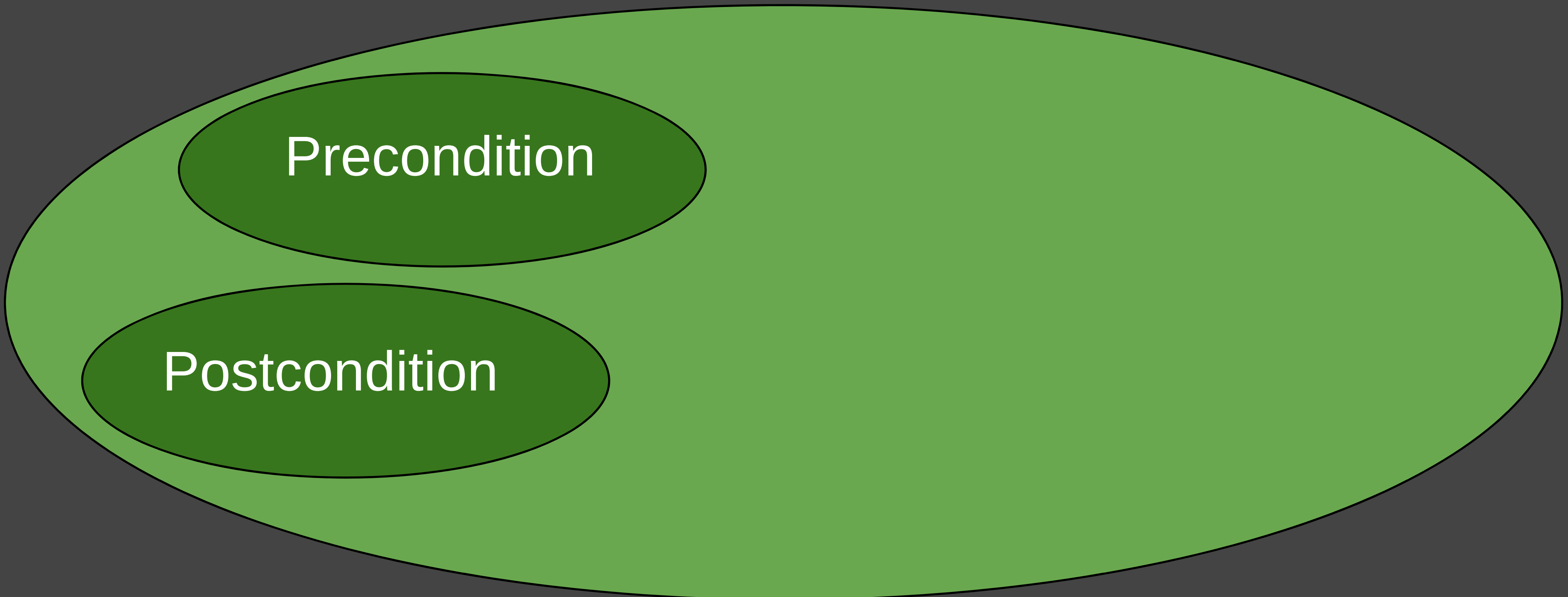
- Reusable Software Research Group at Clemson
- Integrated Programming, Specification, and Proof Language
- Full end-to-end verification
    - Scalability
    - Performance
- Isabelle Backend

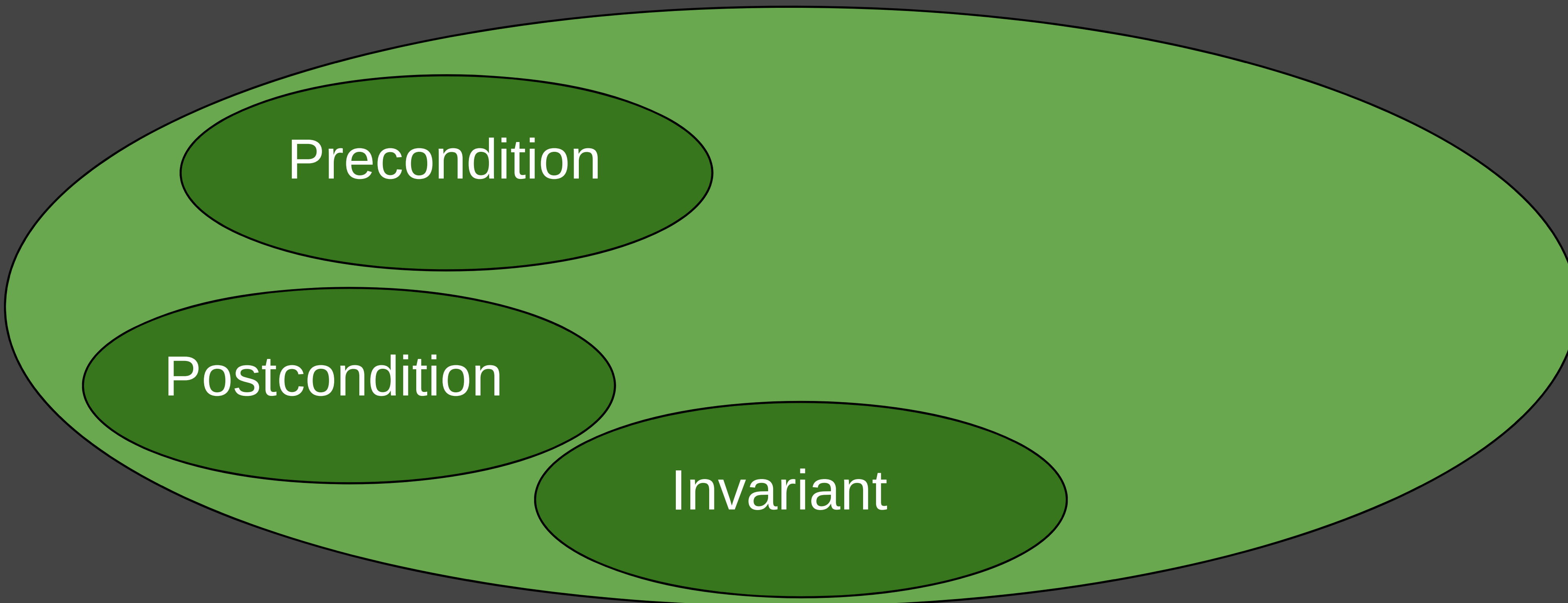
cs.clemson.edu/~resolve

Proof Checkers in a Verification System

PROOF OBLIGATIONS


Precondition


Precondition
Postcondition


Precondition
Postcondition
Invariant

Enhancement for Stacks

```
Enhancement Flipping_Capability for Stack_Template;

    Operation Flip( updates S : Stack );
        ensures S = Rev( #S );

end Flipping_Capability;
```

Implementation of Flipping

```
Realization Obvious_Flipping_Realization for
      Flipping_Capability of Stack_Template;

   Procedure Flip ( updates S : Stack );
      Var Next_Entry : Entry;
      Var S_Flipped : Stack;

      While ( Depth( S ) /= 0 )
         changing S, Next_Entry, S_Flipped;
         maintaining #S = Rev( S_Flipped ) o S;
         decreasing |S|;
      do
         Pop( Next_Entry, S );
         Push( Next_Entry, S_Flipped);
      end;

      S :=: S_Flipped;
   end Flip;
end Obvious_Flipping_Realization;
```

Verification Condition

((|S| <= Max_Depth) and (S = (Rev(?S_Flipped) o ??S) and
(|??S| /= 0 and ??S = (<?Next_Entry> o ?S))))

============================>

(Rev(?S_Flipped) o ??S) =
(Rev(<?Next_Entry> o ?S_Flipped) o ?S)

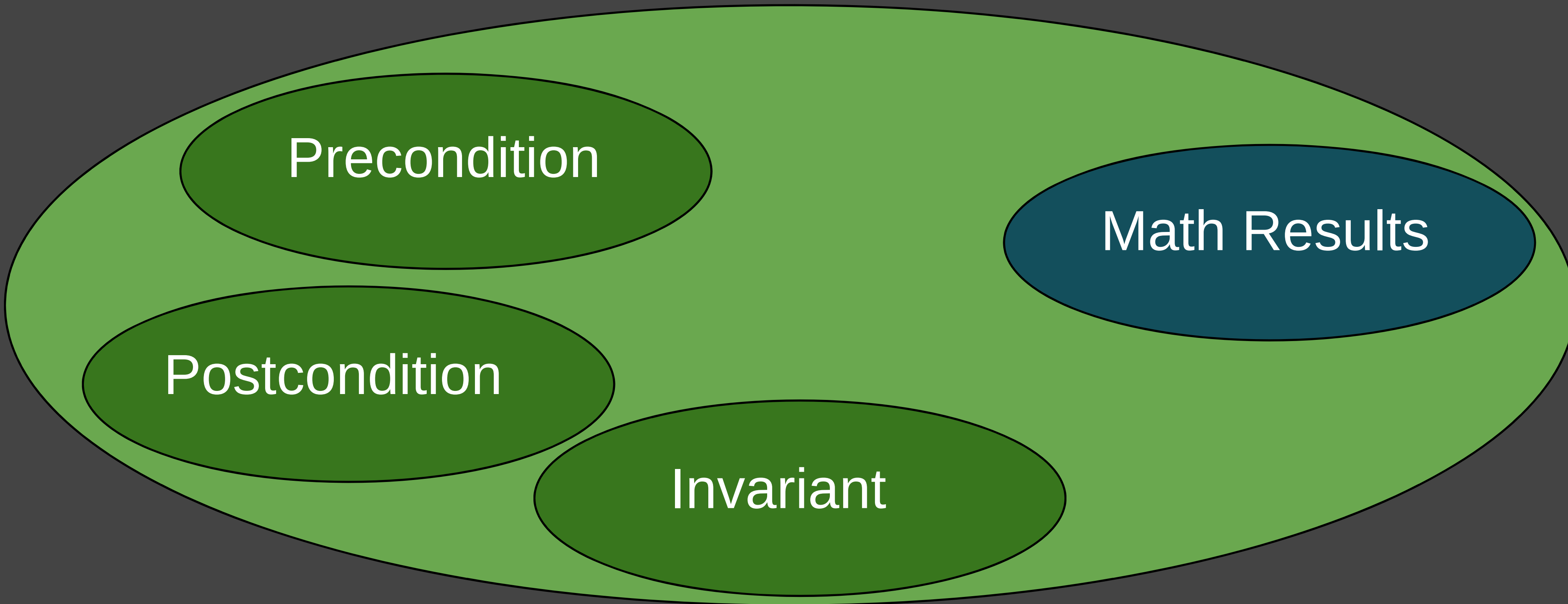
A little help

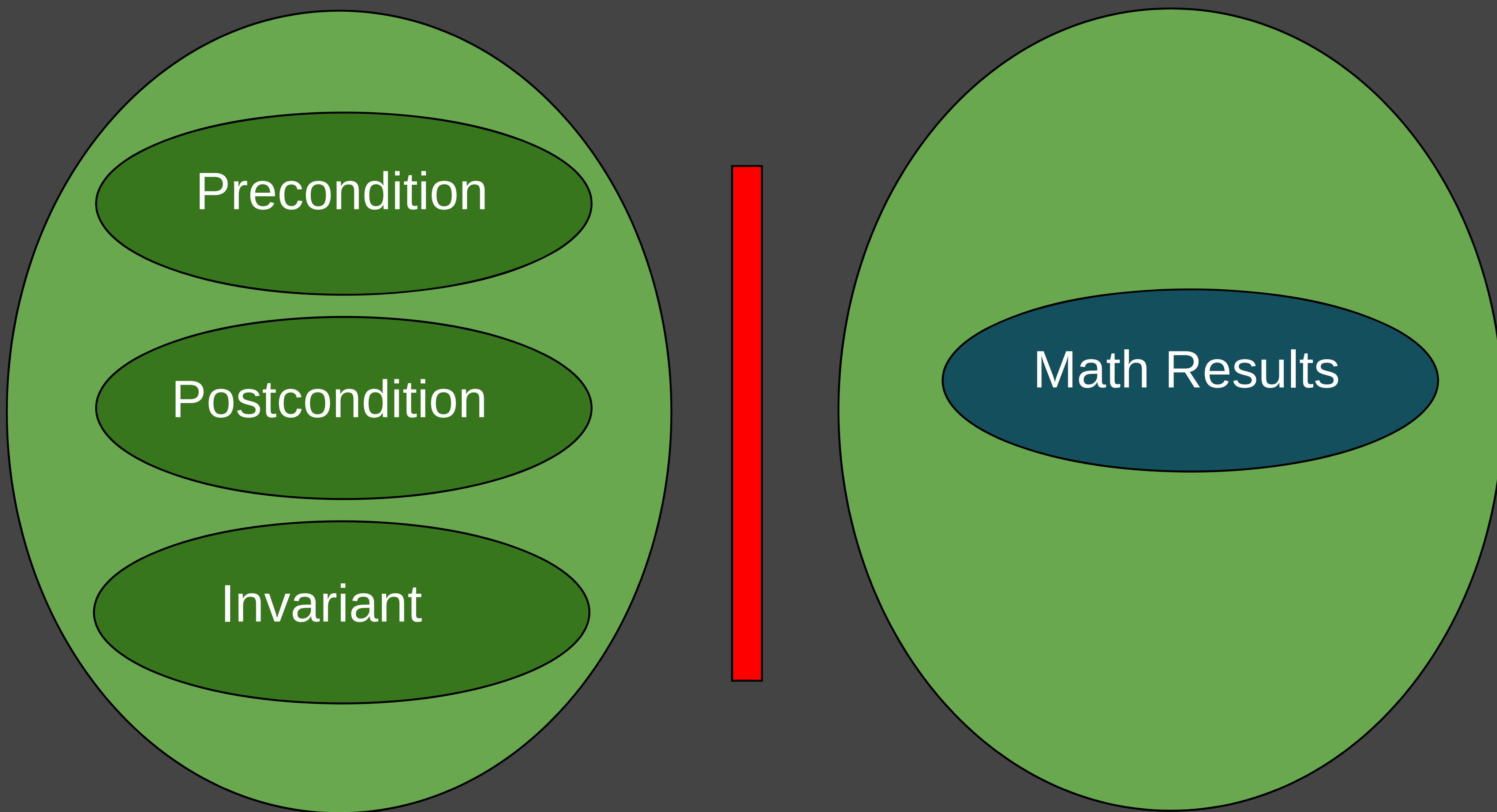
Theorem 1:

$\forall \alpha$: Str(E), $\forall x$: E, $(\alpha \circ \langle x \rangle)^{Rev} = (\langle x \rangle \circ \alpha^{Rev})$

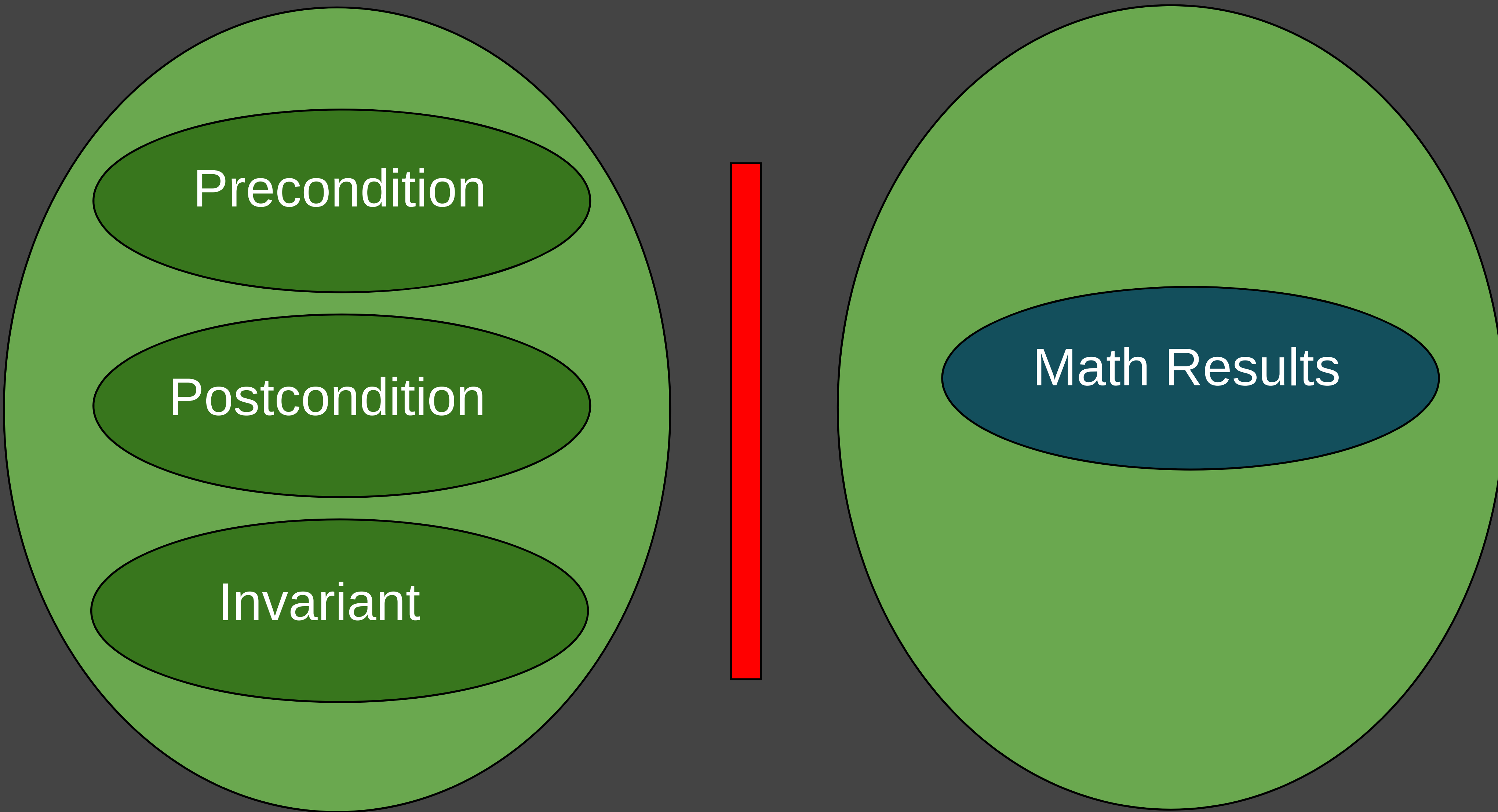
Theorem 2:

Is_Associative($\circ$)

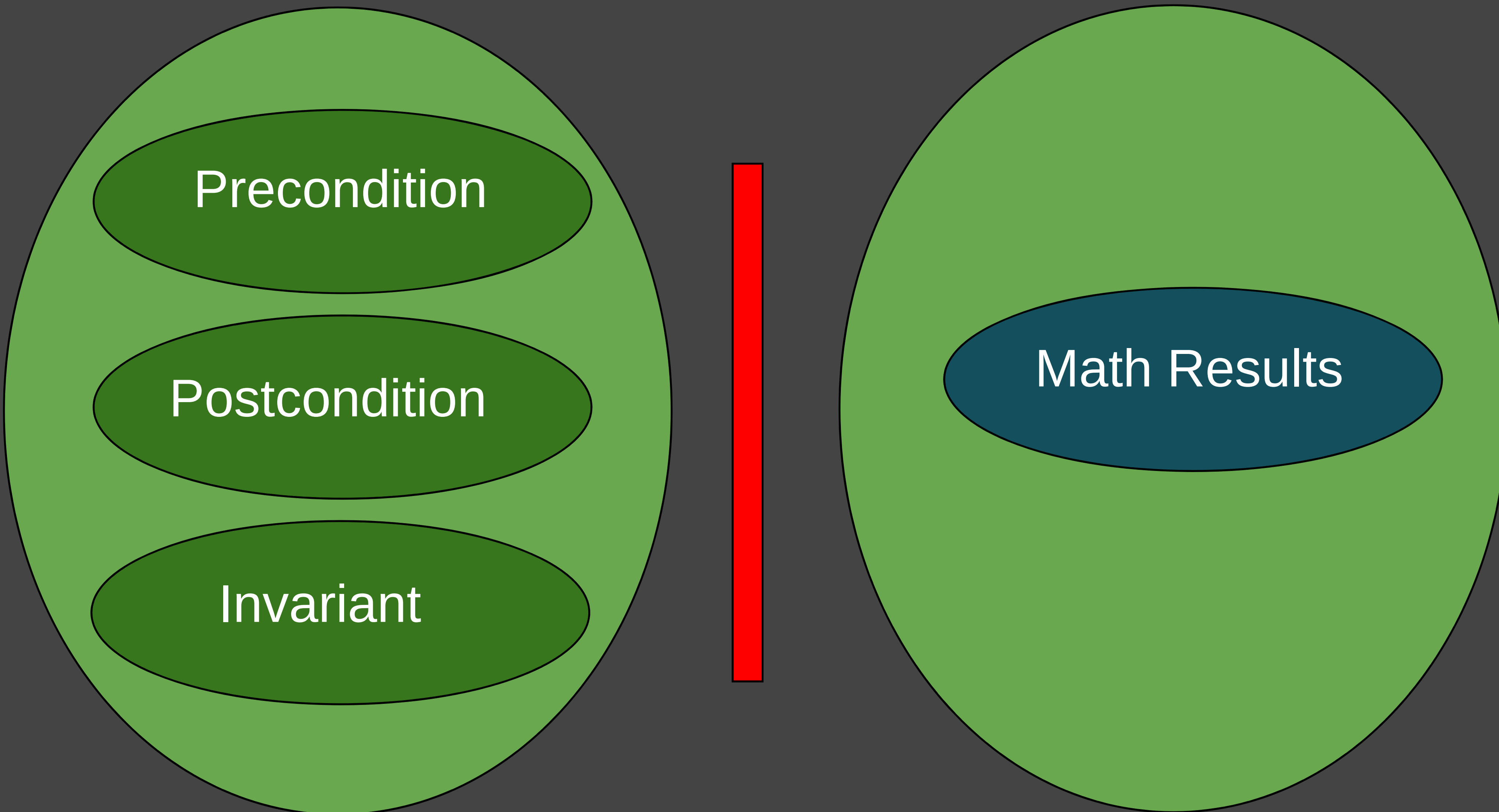

Precondition
Math Results
Postcondition
Invariant


Precondition
Postcondition
Invariant
Math Results

Automated Prover



Automated Prover



User Provided Proof +
Proof Checker

Verification System

"Requiring programmers to engage in a fine level of proof activity is unlikely to lead to wide-spread verification .... [T]he limitations of automated theorem proving often require substantial human intervention."

Verification System

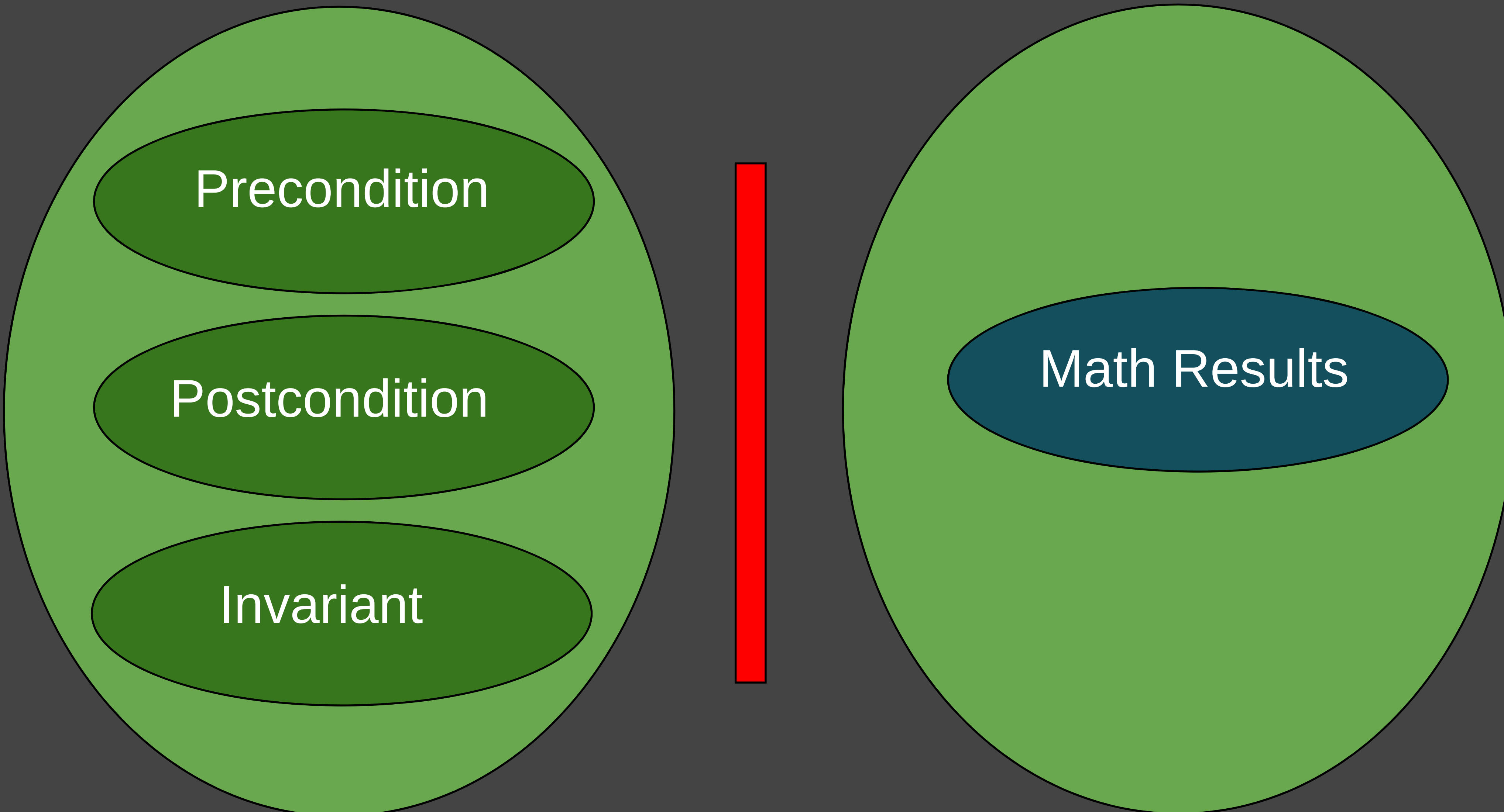
"Requiring programmers to engage in a fine level of proof activity is unlikely to lead to wide-spread verification .... [T]he limitations of automated theorem proving often require substantial human intervention."

Clear division between verification conditions and math results.

Rethink the latter as a job for trained mathematicians.

Requirements for such a Proof Checker

Automated Prover



User Provided Proof +
Proof Checker

Reusability

Programming Language

- Abstraction
- Modules
- Interfaces
- Readability

Proof Language

Reusability

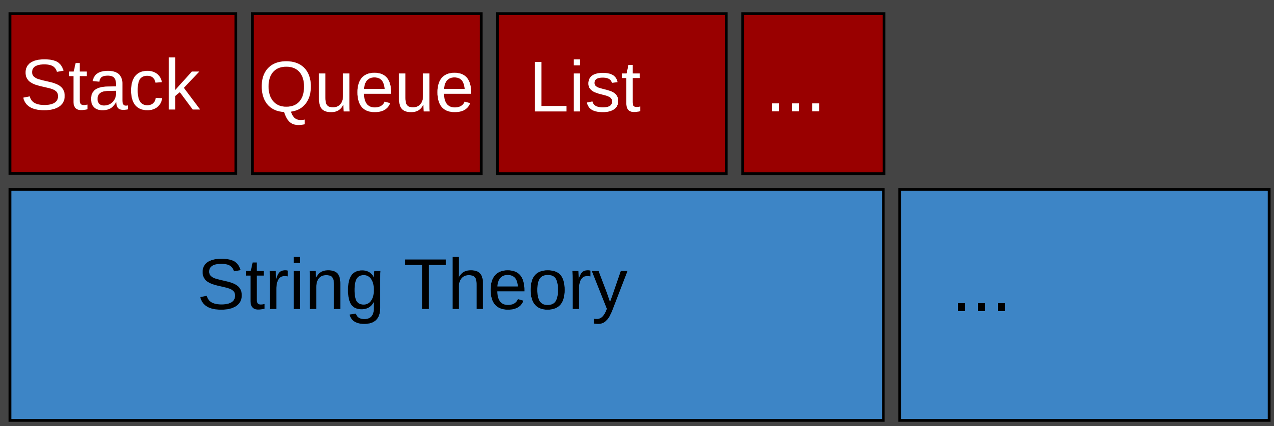
Programming Language

- Abstraction
- Modules
- Interfaces
- Readability

Proof Language

- Abstraction
- Modules
- Interfaces
- Readability

Abstraction and Modules



Consumers of Theories

- Proof Checker
- Automated Prover
- Mathematicians
- Programmers

Précis vs. Proof Units

Header file for theories.

Précis vs. Proof Units

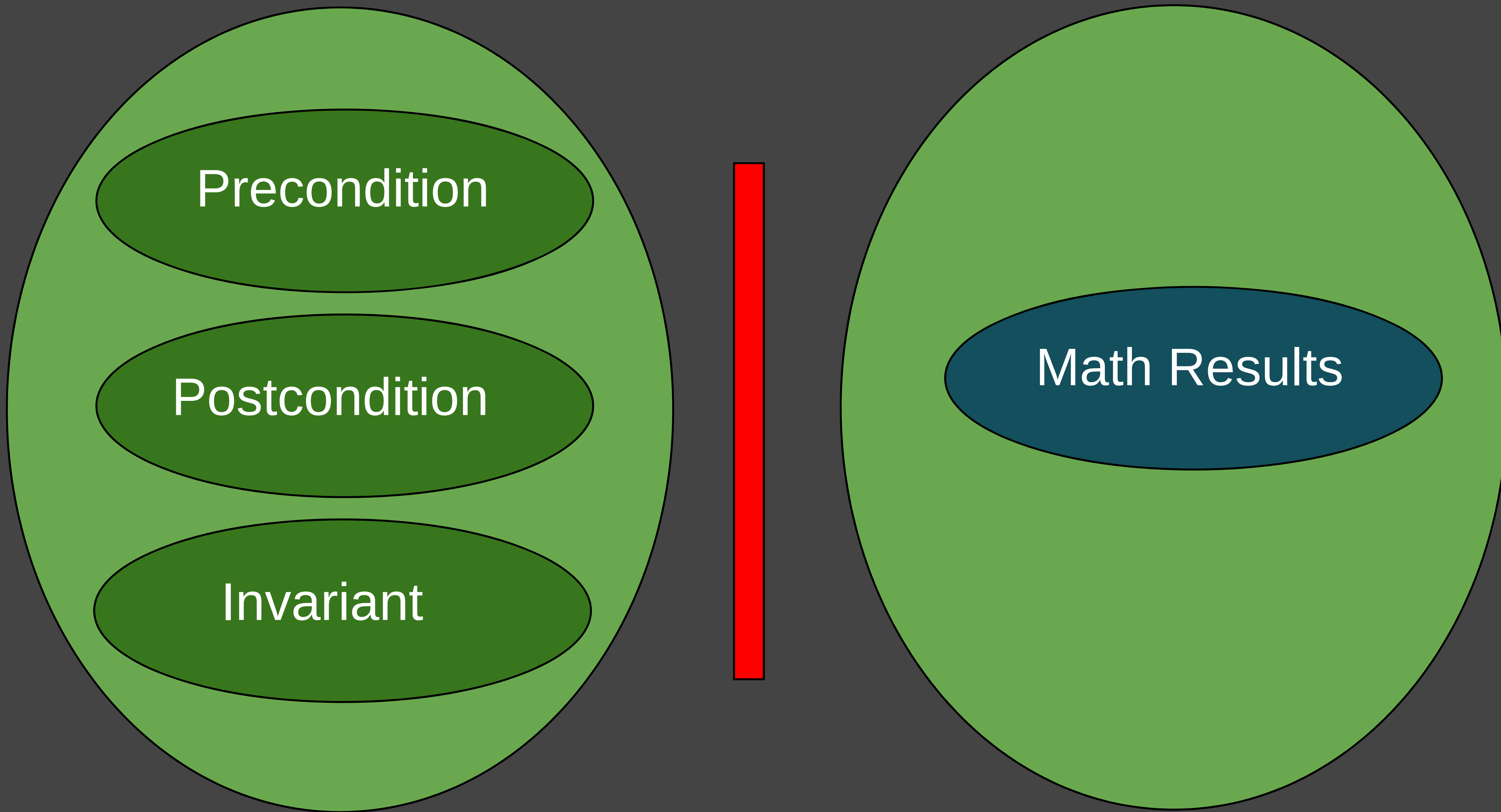
```
Précis Natural_Number_Theory;
uses Basic_Function_Properties,
    Monogenerator_Theory...

    Inductive Definition on i : N of
            (a : N) + (b) : N is
        (i) a + 0 = a;
        (ii) a + suc(b) = suc(a + b);

    Theorem N1:
        Is_Associative( + );

    ...

end Natural_Number_Theory;
```

Précis vs. Proof Units

```
Précis Natural_Number_Theory;
uses Basic_Function_Properties,
    Monogenerator_Theory...

    Inductive Definition on i : N of
            (a : N) + (b) : N is
        (i) a + 0 = a;
        (ii) a + suc(b) = suc(a + b);

    Theorem N1:
        Is_Associative( + );

    ...

end Natural_Number_Theory;
```

```
Proof unit
Natural_Number_Theory_Proofs
    for Natural_Number_Theory;
    Uses ...

    Proof of Theorem N1:
        Goal for all k, m, n: N,
            k + (m + n) = (k + m) + n;
        Definition S1: Powerset(N) =
            {n : N, for all k, m : N,
            k + (m + n) = (k + m) + n};

    ...
```

Automated Prover



User Provided Proof + Proof Checker

Popular Proof Checkers

Isabelle [2]

```
lemma assumes AB:
    "large_A /\ large_B"
    shows
        "large_B /\ large_A"
        ( is "?B /\ ?A" )
    using AB
proof
    assume "?A" "?B"
    show ?thesis ..
qed
```

Coq [1]

```
Variables A B C : Prop.

Lemma and_commutative :
        (A /\ B) -> (B /\ A).
    intro.
    elim H.
    split.
    exact H1.
    exact H0.
Save.
```

Mathematical Proof

Supposition k, m: N

Goal k + (m + 0) = (k + m) + 0

k + (m + 0) = k + m

by (i) of Definition +

k + m = (k + m) + 0

by (i) of Definition +

Deduction if k $\in \mathrm{N}$ and $\mathrm{m} \in \mathrm{N}$ then

k + (m + 0) = (k + m) + 0

[ZeroAssociativity] For all k: N, for all m: N,

k + (m + 0) = (k + m) + 0

by universal generalization

RESOLVE Proof Language

```
Supposition k, m: N;
    Goal k + (m + 0) = (k + m) + 0;
    k + (m + 0) = k + m
                                        by (i) of Definition +;
    k + m = (k + m) + 0
                                        by (i) of Definition +;
  Deduction if k is_in N and m is_in N then
      k + (m + 0) = (k + m) + 0;

  [ZeroAssociativity] For all k: N, for all m: N,
      k + (m + 0) = (k + m) + 0
                                 by universal generalization;
```

Demo

```
Corollary Identity: a : N and
    a + 0 = a;

Proof of Theorem Nothing:
    Supposition k, m: N;
        (k + m) + 0 = k + m
            by Corollary Identity & equality;
    Deduction if k is_in N and
        m is_in N then
        (k + m) + 0 = k + m;
QED
```

Demo

```
Corollary Identity: a : N and
    a + 0 = a;

Proof of Theorem Nothing:
    Supposition k, m: N;
        (k + m) + 0 = m + 0
            by Corollary Identity & equality;
    Deduction if k is_in N and
        m is_in N then
        (k + m) + 0 = k + m;
QED
```

```
Error: Simple.mt(10):
    Could not apply substitution to the
justified expression.
    (k + m) + 0 = m + 0
            by Corollary Identity & equality;
```

Demo

```
Corollary Identity: a : N and
    a + 0 = a;

Proof of Theorem Nothing:
    Supposition k, m: N;
        (k + m) + 0 = k + m
              by Corollary Identity & or rule;
    Deduction if k is_in N and
        m is_in N then
        (k + m) + 0 = k + m;
QED
```

```
Error: Simple.mt(10):
    Could not apply the rule Or Rule to
the proof expression.
    (k + m) + 0 = k + m
              by Corollary Identity & or rule;
```

Conclusions

- A clearer distinction is required between those proof obligations that we expect to be dispatched by an automated prover, and those for which we intend to furnish a proof.

- Programmers should not be required to provide proofs.

- Robust mathematical library of theories is required.

- Techniques from programming languages should be applied to mitigate the complexity of such theories.

References

[1] G. Huet, G. Kahn, and C. Paulin-Mohring, “The Coq Proof Assistant: A Tutorial.” INRIA, 2004, pp. 3-18; 45-47.

[2] T. Nipkow. “A Tutorial Introduction to Structured Isar Proofs,” http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle/doc/isar-overview.pdf.