Proof Automation in Constructive Type Theory

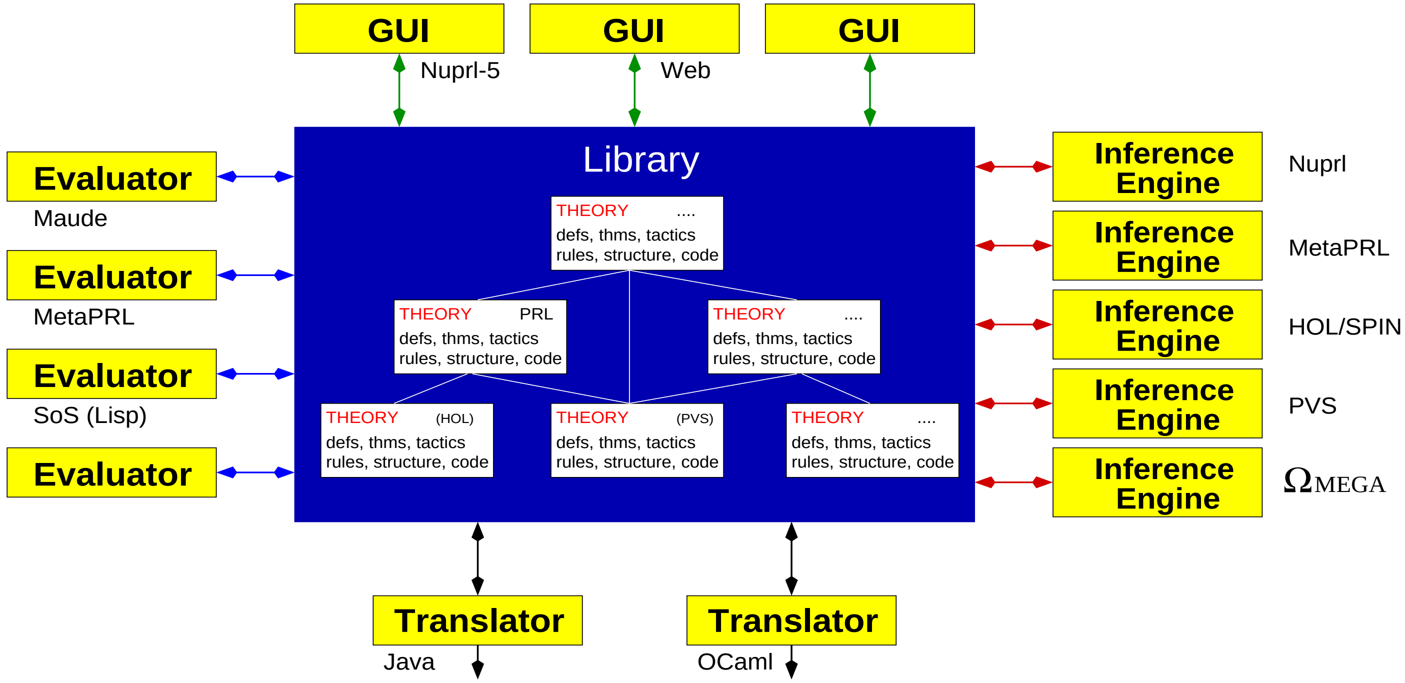
Christoph Kreitz

Nuprl's Type Theory

- **Logic for constructive reasoning**
- **Open-ended, expressive data type system**
  - Function, product, disjoint union, Π- & Σ-types, atoms $\leadsto$ programming
  - Integers, lists, inductive types $\leadsto$ inductive definition
  - Propositions as types, equality type, void, top, universes $\leadsto$ logic
  - Subsets, subtyping, quotient types $\leadsto$ mathematics
  - (Dependent) intersection, union, records $\leadsto$ modules, program composition

  New types can be added as needed
- **Uniform internal notation + "free syntax"** $\leadsto$ display forms
- **Refinement calculus**
  - Top-down sequent calculus $\leadsto$ interactive proof development
- **User-defined extensions possible**

The Nuprl System

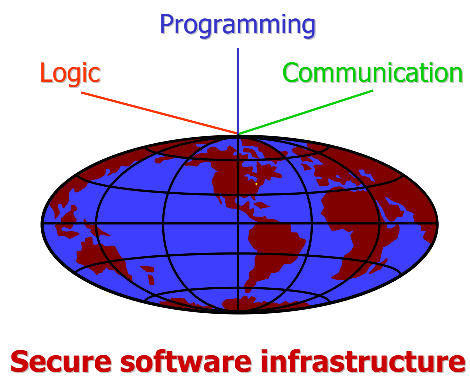


- **Interactive proof development system**
  - Supports constructive proofs and program extraction
  - Some automation by tactics and two decision procedures
  - Flexible definition mechanism with customizable term display
- **Open architecture supports cooperation**
  - Collection of cooperating processes
  - Centered around a common knowledge base
  - Connection to external systems possible (MetaPRL, JProver)

The Need for Proof Automation

- **Nuprl successful in many applications**
  - Verification of communication protocols
  - Optimization of Ensemble protocol stacks
  - Formal design of adaptive systems

    ⋮



. . . but automatic support still too weak

- How to solve conceptually simple proof goals automatically?
- How to decide well-understood problem domains effectively?

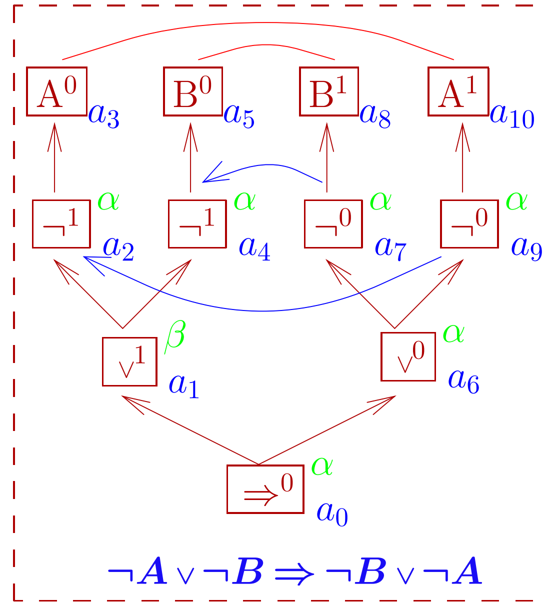
. . . while there are many successful proof mechanisms

- First-order theorem proving
- Strategies for inductive theorem proving
- Proof planning on the meta-level
- Decision procedures for certain application domains
- Model checking

  ⋮

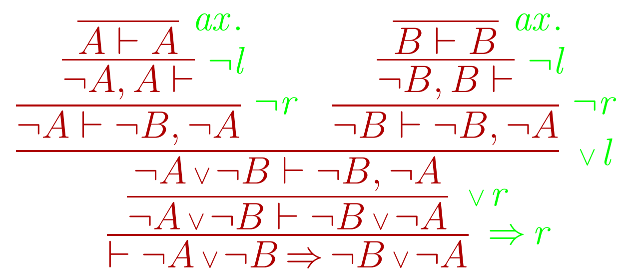
JProver

- **Complete theorem prover for first-order intuitionistic logic**

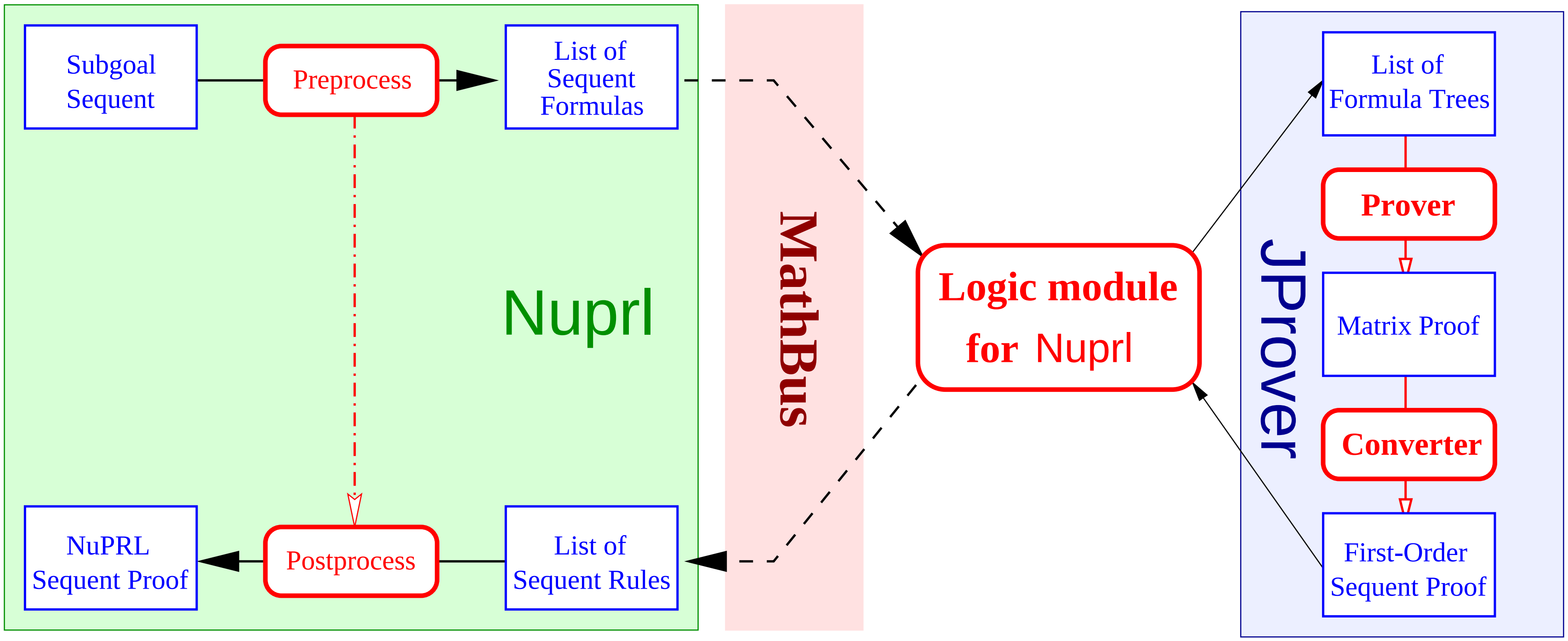


- **Proof Strategy:**
  - Augment formula tree by tableaux types, and polarities
  - Identify connections between leaves (same literal, opposite polarity)
  - Paths through matrix-representation must have complementary connection
  - Validity check reduced to path checking + unification

- **Integration Issues:**
  - Relation between matrix proof and sequent proof
  - Connection between JProver and Nuprl system



INTEGRATING Jprover AND Nuprl



- **Connect as external proof engine**
  – Code module for communicating with Nuprl
  + Logic module for interpeting Nuprl formulas
  + Conversions: sequents $\mapsto$ formulas, matrix proof $\mapsto$ sequent proof
- **Possible improvements:**
  – Processing type information during unification
  – Multiplicities: when to stop creating instances of quantified formulas
  – Adapting problem reduction techniques from classical provers

Can the method for integrating JProver be adapted to other ATP techniques?

- **Implementation as independent proof engine**
  - Use existing procedure in "foreign" prover
  - or reimplementation as MetaPRL code module
- **Explore semantical link to type theory**
- **Build separate module for integration**
  - Code for communicating with Nuprl's library
  - Conversions between different term/proof structures
    or "trusted mode" if proof has no computational content

Some mechanisms may require a different approach

Inductive Theorem Proving with Rippling/Coloring

- **Annotated rewriting for induction step** Bundy, Hutter, ...
- **Proof Strategy:**
  - Use domain-specific rewrite rules (defining equations)
  - Annotate induction hypothesis and conclusion
  - Rewriting of conclusion and hypothesis must reduce measure and result in the same term

$$\boxed{C} \overset{R}{\longmapsto} C_0 \overset{R}{\longmapsto} \ldots \overset{R}{\longmapsto} C_i \overset{R}{\longmapsto} \ldots \overset{R}{\longmapsto} C_n \overset{R}{\longmapsto} \boxed{H}$$

rippling *reverse rippling*

- **Integration issues**
  - Relation between rewrite steps and logical inferences
  - Rewriting weak when quantifiers are involved (logically incomplete)

Integration of Rippling/Coloring

? **Use Clam system as external proof planner**

– Clam guides proof tactics in Oyster, a variant of PeaRL

– Convert rewrite sequence into equality substitutions or implications

? **Extended rippling strategy as Nuprl tactic** Pientka ...

– Analyze proof goal on meta-level
using limited logical decomposition with meta-variables for quantifiers

– Use rippling sequence to generate list of inference steps

Some successful experiments but still logically incomplete

?! **Integrate into JProver**

– Weaken complementarity to allow extending unification by rippling

– Tailor path-checking to check orthogonal connections first

– Add constraint c if proof part fails and check validity for $\neg c$

– Adapt JProver's conversion mechanism

Logically complete

Theoretically explored but not yet implemented

Proof Planning

- **Automatically develop and refine proof sketch**
- **Proof Strategy:** Clam, Ωmega ...
  - Provide plan operators (actions) that specify macro steps (tactics)
    Action = premise + conclusion + application constraint + proof schema
  - Use STRIPS-style planning mechanisms to develop high-level proof plan
  - Refine proof plan during execution
- **Integration issues**
  - Relation between proof plan and primitive inferences/tactics
  - Combining planning with domain knowledge (definitions/lemmata)
  - Expressing domain-specific forms of reasoning as plan control knowledge
  - How to get the relevant knowledge from the library to the planner?
  - How to integrate constraint solving?
- **Proof planning is still very limited**
  - Specialized proof planners not widely applicable

  Could we use more efficient generic planning techniques instead?

Decision Procedures

- **Decide problems in limited application domain**
  - Equality reasoning
  - Congruence closure
  - Subsets of arithmetic
  - List / Tree / Graph / Array theories

    ⋮

- **Proof Strategy:**
  - Translating problem into different problem domain
  - Use well-known decision algorithms

- **Integration issues:**
  - Can we connect only to the refiner of PVS, SVC, etc.?
  - Can the result be trusted (is it consistent with type theory)?
  - What is the constructive content of the proof?
  - Which subterms have to be proven wellformed?
  - Cooperating decision procedures a'la Nelson/Oppen or Shostak?

Model Checking

- **Explore state transition graph to find countermodels**
- **Proof Strategy:**
  - Express system as (finite but huge) state transition graph M and system specification F in temporal logic
  - Explore graph to find all states s such that $M, s \models F$.
  - Return **true** or countermodel for F
  - State explosion solved by problem-reduction (symmetry, BDD's, ...)
- **Very efficient**
  - Successfully used in checking hardware system specifications
  - Applies also to software: separate "finite" components from loops STeP ...
- **Integration as trusted external prover?** SMV ...
  - Convert representation of software in ITT into temporal logic + computation tree logic (= propositional logic + path-temporal operators)
  - Constructive content of a proof? — well-formedness issues?

or conversion into propositional $\leadsto$SAT problem?

Other Techniques

- **Patching faulty proofs**
- **General rewriting and narrowing**
- **Computer Algebra (via proof planning)**
- **Distributed & cooperating proof engines**
- **Machine learning**

⋮