CS 671 Automated Reasoning

Meta Reasoning
Object Level versus Meta Level

- **Object level**: language for formalizing concepts
  - Concrete type theoretical expressions: $x$, $2$, $2^x$, $\lambda x.2^x$, ... Always a formal language

- **Meta level**: describe object level from the outside
  - Term language: “$\lambda x.t$ term if $x$ variable and $t$ term”
    - $x$ and $t$ are syntactical meta-variables
  - Substitution: “$x[t/x] = t$ and $y[t/x] = y$ if $x \neq y$”
  - Evaluation and judgments, validity
  - Sequents, proofs, proof rules, tactics, decision procedures, ...
  - Libraries, theorems, abstractions, display forms, ...
  Often semi-formal: English augmented with formal text
Renaming of bound variables does not change meaning

All Nuprl tactics are correct

Arith is correct
  - An arithmetic sequent $F$ is valid iff the corresponding labelled graph has positive cycles

A first-order formula $F$ is valid iff JProver can prove it
  - $F$ has a sequent proof iff there is a matrix proof for $F$

The algorithm extracted from the proof of intsqrt_4adic runs in logarithmic time

If two record types are syntactically equal up to reordering of labels then they are semantically equal wrt. $\equiv$

$F$ is provable if a certain syntactic transformation of $F$ is

If $F$ has a certain form then tactic tac will always prove it

Meta-reasoning can simplify proof tasks significantly
ML: meta-language as programming language

Express object language as (abstract) data type

```
abstype var = ...
abswtype term = (tok # parm list) # bterm list
and bterm = var list # term
with mk_term (opid,parms) bterms = abs_term((opid,parms),bterms)
and dest_term t = rep_term t
and mk_bterm vars t = abs_bterm(vars,t)
and dest_bterm bt = rep_bterm bt
```

Express proofs and tactics as data types

```
abstype declaration = var # term
lettype sequent = declaration list # term;;
abswtype proof = (declaration list # term) # rule # proof list
with mk_proof_goal decs t = abs_proof((decs,t), \[])
and refine r p = let children = deduce_children r p
and validation = deduce_validation r p
  in children, validation
and hypotheses p = fst (fst (rep_proof p))
and conclusion p = snd (fst (rep_proof p))
and refinement p = fst (snd (rep_proof p))
and children p = snd (snd (rep_proof p))
lettype validation = proof list -> proof;;
lettype tactic = proof -> (proof list # validation);;
```
• Top loops and proof editor reside at meta level

• Object level expressions can be quoted (use C-o)
  – Quoting lifts NUPRL terms to the meta-level
  – Use term editor for editing object level expressions

• Quoted terms can be arguments of ML functions
  – Mostly tactics, computation, decomposition, or substitution

  ... but we can’t reason about the results

  ... and we can’t use ML functions in NUPRL terms
  – can’t define $R_1 \triangleq R_2 \equiv \text{sort-labels}(R_1) = \text{sort-labels}(R_2)$
Can we Reason About the Meta Level?

Meta level of Nuprl is not a logic
... but it has many similarities to type theory

One could use type theory to build a meta-logic

\[
\begin{align*}
\text{Var} & \equiv \text{Atom} \\
\text{Parm} & \equiv \text{Atom} \times \text{Atom} \\
\text{Term} & \equiv \text{rectype Term}=\text{Atom} \times \text{Parm list} \times (\text{Var list} \times \text{Term}) \text{ list}
\end{align*}
\]

\text{mk\_term} \text{ opid parms bterms } \equiv < \langle \text{opid,parms}, \text{bterms} \rangle, \text{bterms} >

\text{mk\_lambda \ var \ t } \equiv \text{mk\_term} "\text{lambda}" [] [[\text{var} \ t ]]

\text{Declaration} \equiv \text{Var} \times \text{Term}

\text{Sequent} \equiv \text{Declaration list} \times \text{Term}

\text{Proof} \equiv (\text{Declaration list} \times \text{Term}) \times \text{Rule} \times \text{Proof list}

But that involves a lot of double work

- All meta-level constructs (evaluation, tactics, ...) need to be lifted
- Meta-logic is part of a different (duplicate) object logic as it does not connect to the logic in which it is defined
- We need to formalize the meta logic of that logic as well
How can we reduce double work?

- **Meta-Logical Frameworks**
  - Build logic for meta level first
  - Embed object logic into meta logic
  - Easy to build (Isabelle, Elf/Twelf, HOL, ...)
  - Can handle multiple logics
  - Fast construction of theorem proving tools for new logics

- **Reflection**
  - Bring meta-logic back into the object logic
  - Reasoning about capabilities of its own meta-logic
  - Replace execution of complex tactics by applying meta-theorems
  - More complex but much more powerful
Logical Frameworks

• Simple logic and proof environment for meta-level
  - Higher order logic of $\forall \Rightarrow$ together with $\lambda$-calculus
  - Fast mechanisms for matching, unification, rewriting

• Represent generic proof theory
  - Terms, sequents, proofs, rules, tactics, ...
  - Prove generic meta-theorems

\[
\forall A, B, C, T_1, T_2. \text{is_rule}(A, B \vdash C) \Rightarrow \text{is_thm}(\vdash T_1) \Rightarrow \text{is_thm}(\vdash T_2) \\
\Rightarrow \text{match}(A, T_1, \sigma) \Rightarrow \text{match}(B, T_2, \sigma) \Rightarrow \text{is_thm}(\vdash \sigma(C))
\]

  - Build fast generic proof tactics

• Define object logic as (inductive) data types
  - Concrete term language, specific rules
  - Prove that specific logic fits generic theory
  - Build proof tactics specialized to object logic
**Reflection**

- **Represent meta-logic** as **Nuprl expressions**
  - Data types for terms, sequents, proofs, rules, tactics, ... 
  - $\lambda$-expressions for substitution, evaluation, refinement, ... 
  - Informally prove isomorphism $\text{Term} \equiv \text{term}$, $\text{Proof} \equiv \text{proof}$, ...

- **Express object logic** in represented meta logic
  - $\lambda$-expressions for building concrete terms and rules 
  - Display forms + color to make embedded logic look like object logic

- **Build hierarchy of levels**
  - Level $i$ is meta level for level $i+1$

- **Reflection rule** links meta level to object level
  
  $H \vdash_{i+1} A$ \hspace{1cm} by reflection $i$

  $[H] \vdash_i \exists p: \text{Proof}_i. \text{goal}(p) = [A]$

  - Use same reasoning apparatus for object and meta level reasoning

---

**Theoretically clean but impractical**