CS 671 Automated Reasoning

Reflection
**Reflection – basic methodology**

- **Represent object and meta level in type theory**
  - Represent meta-logical concepts as NUPRL terms
  - Express specific object logic in represented meta logic
  - Build hierarchy: level $i$ contains meta level for level $i+1$
  - Reasoning about both levels from the “outside”

- **Link object logic and meta-logic**
  - Embed object level terms using quotation
  - Embed object level provability using reflection rule

- **Use same reasoning apparatus for object and meta level**
Reflection, technically (1)

- Represent object level terms
  \[
  \text{Term} \equiv \text{rectype Term} = \text{Atom} \times \text{Parm list} \times (\text{Var list} \times \text{Term}) \text{ list}
  \]
  \[
  x \equiv \text{"variable", [x:v]} > []
  \]
  \[
  \lambda x.t \equiv \text{"lambda", []} > [[x], t]
  \]
  \[
  (f \ t) \equiv \text{"apply", []} > [[]], f;,[], t]
  \]

- Represent meta level operators
  \[
  \text{subst : Term -> Var -> Term -> Term}
  \]
  \[
  \text{evalto: Term -> Term}
  \]
  \[
  \text{canonical: Term -> B}
  \]
  \[
  \text{in: Term -> Term -> P}
  \]

- Represent the proof theory
  \[
  \text{Sequent} \equiv \text{(Var \times Term)} \text{ list} \times \text{Term}
  \]
  \[
  \text{Proof} \equiv \text{Dequent} \times \text{Rule} \times \text{Proof list}
  \]
**Reflection, technically (2)**

- **Prove semantical relationships**
  
  Term \( \triangleq \) term
  
  \([t]\) in \([T]\) \(\triangleq t \in T\)
  
  Proof \(\triangleq\) proof
  
  \([t_1]\) evalto \([t_2]\) \(\triangleq t_1 \downarrow t_2\)
  
  \(\exists p : \text{Proof}.\text{goal}(p) = [H \vdash A]\) \(\triangleq H \vdash A\) is valid
  
- **Add reflection rule**
  
  \(H \vdash_{i+1} A\) by reflection \(i\)
  
  \(\vdash_i \exists p : \text{Proof}_i. \text{goal}(p) = [H \vdash_{i+1} A]\)
  
- **Prove that reflection does not change logic**
  
  - If a sequent \(s\) is provable then it is provable without reflection

Can we use naive reflection?

\[ H \vdash A \quad \text{by reflection} \]
\[ H \vdash \exists p:\text{Proof.} \; \text{goal}(p) = [\vdash A] \]

This would enable us to prove

\[ \vdash \neg (\exists p:\text{Proof.} \; \text{goal}(p) = [\vdash \text{False}]) \]

BY notR

\[ \exists p:\text{Proof.} \; \text{goal}(p) = [\vdash \text{False}] \vdash \text{False} \]

BY reflection

\[ \exists p:\text{Proof.} \; \text{goal}(p) = [\vdash \text{False}] \vdash \exists p:\text{Proof.} \; \text{goal}(p) = [\vdash \text{False}] \]

BY hypotheses

But Gödel’s second incompleteness theorem states

If a consistent, axiomatizable theory \( T \) subsumes arithmetic, then it is impossible to prove the consistency of \( T \) within \( T \)
**Why levels of reflection?**

What if we require all hypotheses to be reflected?

\[ H \vdash A \quad \text{by reflection} \]

\[ \vdash \exists p : \text{Proof}. \; \text{goal}(p) = [H \vdash A] \]

If this rule does not change the logic we should be able to prove

\[ \vdash (\exists p : \text{Proof}. \; \text{goal}(p) = [H \vdash A]) \Rightarrow (H \Rightarrow A) \]

without the reflection rule, which violates Gödel’s theorem.

Adding a reflection rule leads to a hierarchy of proof levels, which may not be closed off proof theoretically. The reflection rule must include indices to separate the levels.

See “Metaprogramming in Nuprl using Reflection” (W. Aitken, PHD Thesis 1994)
Reflection in practice

- **Reflection leads to blow-up of term size**
  - Small terms represented by large tuples

- **Abstractions and display forms can reduce blow-up**
  - Prove laws of reflected concepts and terms
  - Don’t unfold definitions in formal reasoning
  - Use colors in displays to separate levels

- **Substitution and computation remain inefficient**
  - Mechanisms have to be simulated to avoid unfolding terms
  - Can’t use built-in mechanisms
Change the **internal** representation of **Nuprl terms**
- Include quotation level as additional parameter of every term
- All object levels use the same term syntax
  
  \[
  x \triangleq \text{variable} \{x:v, 0:Q\}()
  \]
  \[
  \lambda x.t \triangleq \text{lambda} \{0:Q\}(x.t)
  \]
  \[
  (f \, t) \triangleq \text{apply} \{0:Q\}(f; t)
  \]
  \[
  x \triangleq \text{variable} \{x:v, 1:Q\}()
  \]
  \[
  \lambda x.t \triangleq \text{lambda} \{1:Q\}(x.t)
  \]
  \[
  (f \, t) \triangleq \text{apply} \{1:Q\}(f; t)
  \]
- Some technical subtleties: mixed quotation levels, quoted bindings, ...

Use **built-in** substitution and computation

Extend type theory by **quotation operator** \([ [t] ]\)
- Meaning \([ [t] ]\) of \(t\) is the obvious term of the next quotation level below
  \[
  [[\text{opid}\{p_i:F_i,j+1:Q\}(\text{subterms})]] = \text{opid}\{p_i:F_i,j:Q\}(\text{subterms})
  \]
- Define operators \text{subst}, \text{evalto}, \text{canonical}, \text{in}, ... using \([ [t] ]\)

Reflection of other concepts almost straightforward
Applications

● Improving **proof automation** in theorem proving
  – Enable proofs by syntactical checks

● **Formal proof theory**
  – Elegant accounts of Gödel’s theorems, …

● **Reasoning about program transformations**
  – Optimizations, aspect weaving

● **Reasoning about computational complexity**
  – Complexity classes
  – Resource-bounded logic