CS 671 Automated Reasoning

Reflection

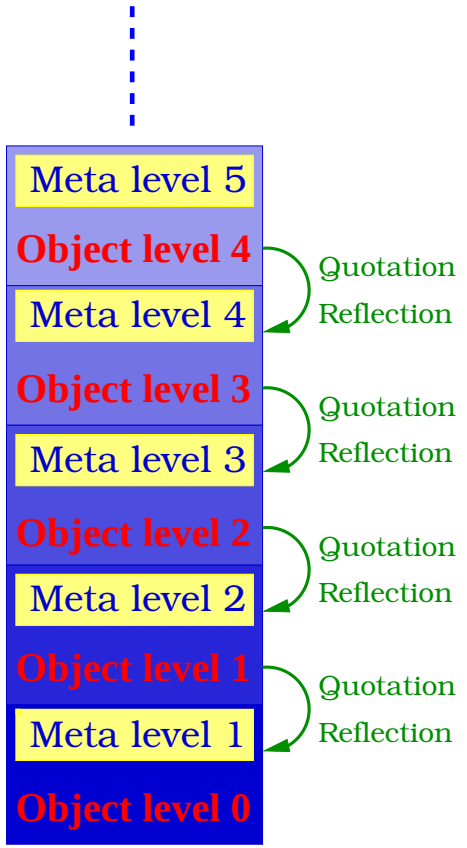
Reflection – basic methodology

- **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

  $\mapsto$ 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**

$$\begin{array}{lcl} \texttt{Term} & \equiv & \texttt{rectype Term = Atom} \times \texttt{Parm list} \times \texttt{(Var list} \times \texttt{Term) list} \\ x & \equiv & \texttt{<<"variable", [}x\texttt{:v]> []>} \\ \lambda x.t & \equiv & \texttt{<<"lambda", []> [[}x\texttt{], }t\texttt{]>} \\ (f\ t) & \equiv & \texttt{<<"apply", []> [[], }f\texttt{;[], }t\texttt{]>} \end{array}$$

- **Represent meta level operators**

```
subst :     Term -> Var -> Term -> Term
evalto:     Term -> Term
canonical:  Term -> 𝔹
in:         Term -> Term -> ℙ
               ⋮
```

- **Represent the proof theory**

$$\begin{array}{lcl} \texttt{Sequent} & \equiv & \texttt{(Var} \times \texttt{Term)list} \times \texttt{Term} \\ \texttt{Proof} & \equiv & \texttt{Dequent} \times \texttt{Rule} \times \texttt{Proof list} \end{array}$$

Reflection, technically (2)

- **Prove semantical relationships**

`Term`	$\doteq$	`term`
$\lceil t \rceil$ `in` $\lceil T \rceil$	$\doteq$	$t \in T$
`Proof`	$\doteq$	`proof`
$\lceil t_1 \rceil$ `evalto` $\lceil t_2 \rceil$	$\doteq$	$t_1 \downarrow t_2$
$\exists$`p:Proof.goal(p) =` $\lceil H \vdash A \rceil$	$\doteq$	$H \vdash A$ is valid

- **Add reflection rule**

$$H \vdash_{i+1} A \quad \textbf{by reflection } i$$
$$\vdash_i \exists \texttt{p:Proof}_i.\ \texttt{goal(p) =} \lceil H \vdash_{i+1} A \rceil$$

- **Prove that reflection does not change logic**
  - If a sequent s is provable then it is provable without reflection

See "*The Semantics of Reflected Proof*", (S. Allen, R. Constable, D. Howe, W. Aitken, 1990)

Why levels of reflection?

Can we use naive reflection ?

$H \vdash A$ **by** `reflection`

$H \vdash$ `∃p:Proof. goal(p) =` $\lceil \vdash A \rceil$

This would enable us to prove

```
⊢ ¬(∃p:Proof.goal(p) = ⌈⊢False⌉)
BY notR
   ∃p:Proof.goal(p) = ⌈⊢False⌉ ⊢ False
   BY reflection
      ∃p:Proof.goal(p) = ⌈⊢False⌉ ⊢ ∃p:Proof.goal(p) = ⌈⊢False⌉
      BY hypotheses
```

But Gödel's second incompleteness theorem states

If a consistent, axiomatizable theory $\mathcal{T}$ subsumes arithmetic, then it is impossible to prove the consistency of $\mathcal{T}$ within $\mathcal{T}$

Why levels of reflection?

What if we require all hypotheses to be reflected?

$H \vdash A$ **by** `reflection`

$\vdash$ `∃p:Proof. goal(p) =` $\lceil H \vdash A \rceil$

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

$\vdash$ `(∃p:Proof. goal(p) =` $\lceil H \vdash A \rceil$`)` $\Rightarrow$ `(`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

Reflecting terms more efficiently (E. Barzilay)

- **Change the internal representation of Nuprl terms**
  - Include quotation level as additional parameter of every term
  - All object levels use the same term syntax

    $x \doteq$ `variable {`x`:v, 0:Q}()`
    $\lambda x.t \doteq$ `lambda {0:Q}(`$x.t$`)`
    $(f\ t) \doteq$ `apply {0:Q}(`$f;t$`)`
    $x \doteq$ `variable {`x`:v, 1:Q}()`
    $\lambda x.t \doteq$ `lambda {1:Q}(`$x.t$`)`
    $(f\ t) \doteq$ `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

    $[[opid\{p_i{:}F_i, j\texttt{+1:Q}\}(subterms)]] = opid\{p_i{:}F_i, j\texttt{:Q}\}(subterms)$
  - Define operators `subst`, `evalto`, `canonical`, `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