Computational Logic

Interactive Theorem Proving

Week 14

Cezary Kaliszyk (VO)
Vincent van Oostrom (PS)

January 20, 2016

Summary

So far

Proof Assistants, HOL Light, $\lambda_{\rightarrow}$, λ_P, λ_2, Curry-Howard, Declarative Proof, Mizar, Proofs about Programs, Extraction

Today

- $\lambda_?$
- Isabelle
- Code Generation

Quiz: $\lambda_?$

- The set of sorts: $\{Prop, Type, Type'\}$
- Pseudo-terms: $T ::= Prop | Type | Type' | Var | (\Pi V : T.T) | (\lambda V : T.T) | T\ T$
- Axiom, Axiom, Var, Weak

$$\frac{}{\vdash Prop : Type} \qquad \frac{}{\vdash Type : Type'} \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$$

- Π: if $(s_1, s_2) \in \{(Type, Type), (Prop, Prop), (Type, Prop)\}$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_2}$$

- λ, app, conv

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : s}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B} \qquad \frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \quad \text{where } B =_\beta B'$$

Quiz: $\lambda_?$

- The combination (Type,Type) forms the function types $A \to B$ for A, B:Type.
  This comprises the unary predicate types and binary relations types: $A \to \mathit{Prop}$ and $A \to A \to \mathit{Prop}$.
  - Also: higher order predicate types like $(A \to A \to \mathit{Prop}) \to \mathit{Prop}$.
  - A Π-type formed by (Type,Type) is always an $\to$-type.
- (Prop,Prop) forms the propositional types $\varphi \to \psi$ for $\varphi, \psi : \mathit{Prop}$; implicational formulas.
  - A Π-type formed by (Type,Type) is always an $\to$-type.
- (Type,Prop) forms the dependent propositional type $\Pi x : A.\varphi$ for A:Type, φ:Prop; universally quantified formulas.

Properties of λ_{HOL} (1/2)

Question: is the type theory λHOL really isomorphic with HOL?
Yes: it is possible to disambiguate the syntax.

- Uniqueness of types
  If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.
- Subject Reduction
  If $\Gamma \vdash M : A$ and $M \to_\beta N$, then $\Gamma \vdash N : A$.
- Strong Normalization
  If $\Gamma \vdash M : A$, then all β-reductions from M terminate.

Properties of λ_{HOL} (2/2)

Decidability Questions:

$$\Gamma \vdash M : \sigma ? \, TCP$$

$$\Gamma \vdash M : ? \, TSP$$

$$\Gamma \vdash ? : \sigma \, TIP$$

For λHOL:

- TIP is undecidable
- TCP/TSP
  - Formally: we introduce a judgement of correct context, algorithm close to λP

λ_{HOL} contains $\lambda 2$ and $\lambda \rightarrow$

if $(s_1, s_2) \in \{(\mathit{Type}, \mathit{Type}), (\mathit{Prop}, \mathit{Prop}), (\mathit{Type}, \mathit{Prop})\}$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_2}$$

This rule allows to form

- $\rightarrow$-types on the Type-level (first copy of $\lambda\rightarrow$)
- $\rightarrow$-types on the Prop-level (second copy of $\lambda\rightarrow$)
- $\Pi\alpha$:Prop.$\alpha\rightarrow\alpha$: polymorphic types on the Prop-level (copy of $\lambda 2$)

HOL in LF (1/2)

```
holtype : type
bool : holtype
fun  : holtype → holtype → holtype                          "1 ⇒ 2"

term  : holtype → type
Abs   : {A,B} (term A → term B) → term (A ⇒ B)               "λ 3"
Comb  : {A,B} term (A ⇒ B) → term A → term B                 "3 ' 4"
equal : {A} term A ⇒ (A ⇒ bool)                              "2 = 3"

thm   : term bool → type                                     "⊢ 1"
REFL  : {A,X:term A} ⊢ X = X
TRANS : {A,X,Y,Z:term A} ⊢ X = Y → ⊢ Y = Z → ⊢ X = Z
MP    : {p,q} ⊢ p = q → ⊢ p → ⊢ q
BETA  : {A,B,F:term A → term B,X:term A} ⊢ (λ F)'X = (F X)
MK_COMB : {A,B, F,G:term A⇒B, X,Y:term A} ⊢ F = G → ⊢ X = Y → ⊢ F'X = G'Y
ABS   : {A,B, S,T:term A → term B}
  ({x: term A} ⊢ (S x) = (T x)) → ⊢ λ S = λ T
DEDUCT_ANTISYM_RULE : {p,q} (⊢ p → ⊢ q) → (⊢ q → ⊢ p) → ⊢ p = q
```

HOL in LF (2/2)

```
extension definition =
   [n: nat] [A: holtype^n → holtype] [a: {T: holtype^n} term (A T)]
      c   : {T} term (A T)
      DEF : {T} ⊢ (c T) = (a T)

extension new_basic_type_definition =
   [n:nat] [A: holtype^n → holtype]
   [P: {T: holtype^n} term (A T) ⇒ bool]
   [w: {T: holtype^n} term (A T)]
   [nonempty: {T: holtype^n} ⊢ (P T) ' (w T)]
      B       : holtype^n → holtype
      abs     : {T} term (B T) ⇒ (A T)
      rep     : {T} term (A T) ⇒ (B T)
      rep_abs : {T} {b: term (B T)} ⊢ (rep T) ' ((abs T) ' b) = b
      abs_rep : {T} {a: term (A T)} ⊢
                        (P T) ' a = ((abs T) ' ((rep T) ' a) = a)
```

Isabelle

Kernel

- Implemented in SML
  - LCF style
  - Makes use of PolyML checkpointing, JIT, nat, ...
- Medium size
  - ($\approx$ 5-10 files)
  - Higher order unification
  - Is code generation part of it?
- Weak type theory
  - Meta-logic Pure
  - Polymorphic types and type classes
  - Explicit connectives: Implication, Equality, Universal quantifier
  - Implicit: Conjunction, Meta-Existence
- Generic theorem prover
  - Larry Paulson: "Isabelle: The Next 700 Theorem Provers"

Object logics

- IFOL
  - FOL
    - ZF
    - LCF (original Edinburgh LCF logic and prover from 1972)
- CTT
  - First version by Martin-Löf in 1971 impredicative
  - After discovery of Girard's paradox later versions predicative
- HOL (minimally different from HOL Light)
- TLA(+) (language for software/hardware specifications)
- ...

Defining a logic

```
typedecl o

axiomatization
  False :: o and
  conj :: "o => o => o" (infixr "&" 35) and
  disj :: "o => o => o" (infixr "|" 30) and
  imp :: "o => o => o" (infixr "-->" 25)
where
  conjI: "P ==> Q ==> P&Q" and
  conjunct1: "P&Q ==> P" and
  conjunct2: "P&Q ==> Q" and
  disjI1: "P ==> P|Q" and
  disjI2: "Q ==> P|Q" and
  disjE: "P|Q ==> (P ==> R) ==> (Q ==> R) ==> R" and
  impI: "(P ==> Q) ==> P-->Q" and
  mp: "P-->Q ==> P ==> Q" and
```

Soundness and Completeness

Is the Isabelle representation of logic correct?

- Each axiom is sound with respect to the truth-table semantics
- Syntactic rule-by-rule translation
  - For each meta-proof there is a corresponding object proof

Completeness

- Object proofs are translated to meta-proofs
  - By induction on the size of the proof-object

Intuitionistic logic with natural deduction

Theory NJ

- First order logic (Prawitz, 1965)
- Combination of forward and backward reasoning

Rules

```
[| P |] ==> [| Q |] ==> [| P&Q |]
[| P&Q |] ==> [| P |]       [| P&Q |] ==> [| Q |]
[| P |] ==> [| P|Q |]       [| Q |] ==> [| P|Q |]
[| P|Q |] ==> ([| P |] ==> [| R |]) ==>
              ([| Q |] ==> [| R |]) ==> [| R |]
([| P |] ==> [| Q |]) ==> [| P-->Q |]
[| P-->Q |] ==> [| P |] ==> [| Q |]
```

Quantifiers

```
(!(y) [| P(y) |]) ==> [| ALL x.P(x) |]
[| ALL x.P(x) |] ==> [| P(a) |]
[| P(a) |] ==> [| EXISTS x.P(x) |]
[| EXISTS x.P(x) |] ==> (!(y)[| P(y) |] ==> [| R |]) ==> [| R |]
```

Constructive Type Theory

Theory CTT

- Extensional version of Martin-Löf Type Theory
- Normally: Typing judgements ($a(...) \in A(...)$)
- But also: being a family of types over A (B(x) type)

Rules

```
[| A type |] ==> [| B type |] ==> [| A+B type |]
[| a: A |] ==> [| B type |] ==> [| inl(a): A+B |]
[| p: A+B |] ==> (!(x)[| x: A |] ==> [| c(x): C(inl(x)) |]) ==>
                 (!(y)[| y: B |] ==> [| d(y): C(inr(y)) |]) ==>
                 [| when(p,c,d): C(p) |]
[| a: A |] ==> (!(x)[| x: A |] ==> [| c(x): C(inl(x)) |]) ==>
               (!(y)[| y: B |] ==> [| d(y): C(inr(y)) |]) ==>
               [| when(inl(a),c,d) = c(a): C(inl(a)) |]
```

Intuitionistic and Classical FOL

Theories IFOL and FOL

- Sequent calculus with sequent variables

Sequents, Thinning, Cut

```
[| $H, P, $G |- $E, P, $F |]
[| $H |- $E, $F |] ==> [| $H |- $E, P, $F |]
[| $H |- $E, P |] ==> [| $H, P |- $E |] ==> [| $H |- $E |]
```

Conjunction and Negation

```
[| $H |- $E, P, $F |] ==> [| $H |- $E, Q, $F |] ==>
                          [| $H |- $E, P&Q, $F |]
[| $H, P, Q, $G |- $E |] ==> [| $H, P&Q, $G |- $E |]
[| $H, P |- $E, $F |] ==> [| $H |- $E, ~P, $F |]
[| $H, $G |- $E, P |] ==> [| $H, ~P, $G |- $E |]
```

Zermelo-Fraenkel set theory

Theory ZF

- Based on FOL
  - Axioms of ZF are complex
- Extensionality is defined using subsets
- Power-set axiom states that $A \in Pow(B) \iff A \subseteq B$
- Limited comprehention using `Collect`
- Replacement axiom separate `Replace`
- Isabelle formalizes schemes using function variables

```
Collect(A,P)== a:A & P(a)
Replace(f,B) == EXISTS a. a:B & c=f(a)
```

Looking at the theory

Isabelle: Contesting for the biggest formal library

HOL + Other Logics

ZF, CTT, TLA+

Efficiency

Object-Logic, but PolyML+JIT, Checkpointing, Limited proof objects

Code Generation

HOL: Big Library + AFP

Math, Protocols, Algorithms, Programs

Isar

Declarative proof, Tactics, Access to ML

Code generation

- HOL (Isabelle) specifications
- can be turned into executable programs
  - Haskell, OCaml, SML, Scala
- Shallow embedding
  - Program semantics generated from equational theorems
  - (relation to higher-order rewriting)

Example (1/2)

```
datatype 'a queue = AQueue 'a list 'a list

definition empty :: 'a queue where
  empty = AQueue [] []

primrec enqueue :: 'a => 'a queue => a' queue where
  enqueue x (AQueue xs ys) = AQueue (x # xs) ys

fun dequeue :: 'a queue => 'a option * 'a queue where
  dequeue (AQueue [] []) = (None, AQueue [] [])
| dequeue (AQueue xs (y # ys)) = (Some y, AQueue xs ys)
| dequeue (AQueue xs []) =
  (case rev xs of y # ys => (Some y, AQueue [] ys))

export-code empty dequeue enqueue in SML
  module-name Example file examples/example.ML
```

```
structure Example = struct

fun foldl f a [] = a
  | foldl f a (x :: xs) = foldl f (f a x) xs;

fun rev xs = foldl (fn xsa => fn x => x :: xsa) [] xs;

fun list_case f1 f2 (a :: lista) = f2 a lista
  | list_case f1 f2 [] = f1;

datatype 'a queue = AQueue of 'a list * 'a list;

val empty : 'a queue = AQueue ([], [])

fun dequeue (AQueue ([], [])) = (NONE, AQueue ([], []))
  | dequeue (AQueue (xs, y :: ys)) = (SOME y, AQueue (xs, ys))
  | dequeue (AQueue (v :: va, [])) =
    let
      val y :: ys = rev (v :: va);
    in
      (SOME y, AQueue ([], ys))
    end;

fun enqueue x (AQueue (xs, ys)) = AQueue (x :: xs, ys);
end; (*struct Example*)
```

Code generation

- Program Refinement
  - Given various equal implementations
  - Selecting Code Equations with an attribute
- Partial programs
  - Explicit “code-abort”
- Datatype refinement
  - Abstract data-types: sets
    - List? RBT?
  - Datatypes with invariants
  - Quotients: λ-terms
- Evaluation techniques
  - In proofs?
- Example: RBT_Set

Verified programs in the large

- Extraction
  - 2000: FTA
    - every non-constant complex polynomial has a root
    - input $x^2 - 2$
    - program approximating $\sqrt{2}$
  - CompCert: optimizing compiler for C
  - SSR: 4 color theorem, Odd order theorem.
- Separation Logic
  - VCG
  - L4-Verified
- Code Generation
  - CeTA

Summary

Today

- Isabelle

Next time

- Presentations