Implementation of a Coherent Logic Prover for Isabelle

Stefan Berghofer
Institut für Informatik
Technische Universität München

joint work with

Marc Bezem
Institutt for Informatikk
Universitetet i Bergen
Roadmap

1. Background

2. Isabelle’s Logic

3. Coherent Logic in Isabelle

4. Conclusion
Background
Isabelle

- Developed (since 1986) by Larry Paulson (Cambridge) and Tobias Nipkow
- Interactive theorem prover
- Logical Framework
  Description of various object logics using a meta logic (Isabelle/Pure)
- Most well-developed object logic: Isabelle/HOL
- Design philosophy
  - Inferences may only be performed by a small kernel ("LCF approach")
  - Definitional theory extension
    New concepts (such as inductive datatypes and predicates) must be defined using already existing concepts.

"The method of ‘postulating’ what we want has many advantages;
they are the same as the advantages of theft over honest toil.
Let us leave them to others and proceed with our honest toil."

Bertrand Russell, Introduction to Mathematical Philosophy
A short history of theorem provers


Simple Theory of Types [Church]

LCF [Scott]

Stanford LCF [Milner]

Edinburgh LCF [Milner]

Cambridge LCF [Paulson]

Nuprl [Constable]

Martin–Löf Type Theory

LF [Harper]

Elf [Pfenning]

Isabelle [Paulson]

Isabelle/HOL [Nipkow]

HOL 88 [Gordon]

HOL90 [Slind]

HOL Light [Harrison]

HOL 98

HOL4

System F [Girard]

Calculus of Constructions [Coquand, Huet]

Coq [Coquand, Paulin, Huet]

LEGO [Pollack]

Epigram [McBride]

Nqthm [Boyer, Moore]

ACL2 [Moore, Kaufmann]

PVS [Owre, Shankar, Rushby]

Twelf [Pfenning, Schürmann]

ACL2

Agda [C. Coquand]

HOL98

HOL4

Automating Coherent Logic, Oslo, 13.6.2008
Architectue of Isabelle/Pure
Theory hierarchy of Isabelle/HOL
Isabelle’s Logic
Formalizing logics in Isabelle

Meta logic Isabelle/Pure

- **Terms:** $t = x | c | \lambda x : \tau. t | t t$
- **Types:** $\tau = \alpha | (\tau_1, \ldots, \tau_n)tc$ where $tc \in \{\Rightarrow, \text{prop}, \ldots\}$
- **Logical operators:**
  - Implication $\implies :: \text{prop} \Rightarrow \text{prop} \Rightarrow \text{prop}$
  - Universal quantifier $\forall :: (\alpha \Rightarrow \text{prop}) \Rightarrow \text{prop}$
  - Equality $\equiv :: \alpha \Rightarrow \alpha \Rightarrow \text{prop}$

Object logic Isabelle/HOL

- **Terms and types:** as in Isabelle/Pure
- **Logical operators:**
  - Truth predicate $[\ldots] :: \text{bool} \Rightarrow \text{prop}$
  - Conjunction $\land :: \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}$
  - Disjunction $\lor :: \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}$
  - Implication $\rightarrow :: \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}$
  - Universal quantifier $\forall :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$
  - Existential quantifier $\exists :: (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}$
Proof representation in Isabelle/Pure

Proofs as $\lambda$-terms

\[
p, q = h \quad \text{Hypothesis}
\]
\[
c_{\alpha \mapsto \tau} \quad \text{Proof constant (reference to axiom / theorem)}
\]
\[
p \cdot t \quad \text{\wedge -elimination}
\]
\[
p \cdot q \quad \text{\rightarrow -elimination}
\]
\[
\lambda x :: \tau. p \quad \text{\wedge -introduction}
\]
\[
\lambda h : \varphi. p \quad \text{\rightarrow -introduction}
\]

Proof checking

\[
\Gamma, h : t, \Gamma' \vdash h : t
\]
\[
\sum(c) = \varphi
\]
\[
\Gamma \vdash c_{\alpha \mapsto \tau} : \varphi_{\alpha \mapsto \tau}
\]
\[
\Gamma \vdash p : \bigwedge x :: \tau. \varphi \quad \Gamma \vdash t :: \tau
\]
\[
\Gamma \vdash p \cdot t : P\{x \mapsto t\}
\]
\[
\Gamma \vdash \lambda x :: \tau. p : \bigwedge x :: \tau. \varphi
\]
\[
\Gamma \vdash p : \varphi \rightarrow \psi \quad \Gamma \vdash q : \varphi
\]
\[
\Gamma \vdash p \cdot q : \psi
\]
\[
\Gamma \vdash \lambda h : \varphi. p : \varphi \rightarrow \psi
\]

Automating Coherent Logic, Oslo, 13.6.2008
Natural deduction calculus [Gentzen 1933]

**Introduction rules**

\[
\frac{P}{P \land Q} (\land I)
\]

\[
\frac{Q}{P \land Q} (\land I)
\]

\[
\frac{P}{P \lor Q} (\lor I_1)
\]

\[
\frac{Q}{P \lor Q} (\lor I_2)
\]

\[
\frac{[P]}{P \rightarrow Q} (\rightarrow I)
\]

**Elimination rules**

\[
\frac{P \land Q}{R} (\land E)
\]

\[
\frac{[P]}{R} (\land E)
\]

\[
\frac{[Q]}{R} (\land E)
\]

\[
\frac{P \lor Q}{R} (\lor E)
\]

\[
\frac{P}{Q} (\rightarrow E)
\]

\[
\frac{P}{\bot} (\bot E)
\]
More rules

\[\frac{}{\neg P} (\neg I)\]

\[\frac{P}{\forall x. P} (\forall I)\]

\[\frac{}{\exists x. P} (\exists I)\]

\[\frac{}{\neg P, P} (\neg E)\]

\[\frac{}{Q} (\forall E)\]

\[\frac{\forall x. P}{P[t/x]} (\exists E)\]

\[\frac{}{\exists x. P, Q} (\exists E)\]

*Variable condition:*

\(\forall I: x\) not free in the assumptions

\(\exists E: x\) not free in \(Q\) or any assumption except \(P\)
Inference rules of Isabelle/HOL

**conjI:** \[ [P] \implies [Q] \implies [P \land Q] \]

**disjI1:** \[ [P] \implies [P \lor Q] \]

**disjI2:** \[ [Q] \implies [P \lor Q] \]

**impl:** \( ([P] \implies [Q]) \implies [P \rightarrow Q] \)

**notl:** \( ([P] \implies [\text{False}]) \implies [\neg P] \)

**allI:** \( (\forall x. [P \ x]) \implies [\forall x. P \ x] \)

**exI:** \( [P \ x] \implies [\exists x. P \ x] \)

**conjE:** \[ [P \land Q] \implies ([P] \implies [Q] \implies [R]) \implies [R] \]

**disjE:** \[ [P \lor Q] \implies ([P] \implies [R]) \implies ([Q] \implies [R]) \implies [R] \]

**mp:** \( ([P \rightarrow Q]) \implies [P] \implies [Q] \)

**FalseE:** \( [\text{False}] \implies [P] \)

**notE:** \( [\neg P] \implies [P] \implies [Q] \)

**spec:** \( [\forall x. P \ x] \implies [P \ x] \)

**exE:** \( [\exists x. P \ x] \implies (\forall x. [P \ x] \implies [Q]) \implies [Q] \)
Unstructured vs. structured proofs

**Theorem** $ex1$: $(\exists x. \forall y. P x y) \rightarrow (\forall y. \exists x. P x y)$

apply (rule impI)
apply (erule exE)
apply (rule allI)
apply (rule exI)
apply (drule spec)
apply assumption
done

**Theorem** $ex2$: $(\exists x. \forall y. P x y) \rightarrow (\forall y. \exists x. P x y)$

proof (rule impI)
  assume $\exists x. \forall y. P x y$ then show $\forall y. \exists x. P x y$
  proof (rule exE)
   fix $x$ assume $h$: $\forall y. P x y$ show $\forall y. \exists x. P x y$
   proof (rule allI)
     fix $y$ from $h$ have $P x y$ by (rule spec) then show $\exists x. P x y$ by (rule exI)
     qed
   qed
  qed
qed
qed
Coherent Logic in Isabelle
General elimination rules

• Since Isabelle is a logical framework, the CL prover should work with any object logic (e.g. HOL, FOL, ZF, ...)
• Can we express CL rules just using the meta logic Isabelle/Pure?

\[ A_1 \land \ldots \land A_m \rightarrow (\exists x_1. B_1^1 \land \ldots \land B_{k_1}^1) \lor \ldots \lor (\exists x_n. B_1^n \land \ldots \land B_{k_n}^n) \equiv \]

\[ A_1 \implies \ldots \implies A_n \implies (\land x_1^1. B_1^1 \implies \ldots \implies B_{k_1}^1 \implies P) \implies \ldots \implies (\land x_n^1. B_1^n \implies \ldots \implies B_{k_n}^n \implies P) \implies P \]

Rules used in the translation

\[ A \equiv (\land B. (A \implies B) \implies B) \]
\[ (A \land B \implies C) \equiv (A \implies B \implies C) \]
\[ ((A \lor B \implies C') \implies C') \equiv (((A \implies C') \implies (B \implies C') \implies C') \]
\[ ((\exists x. P \ x) \implies Q) \equiv (\land x. P \ x \implies Q) \]
Linking external provers to Isabelle

1. Translate Isabelle formula to format understood by prover
2. Write formula to file
3. Call external prover
4. External prover writes result (proof) to log file
5. Reconstruct Isabelle proof from log file

- Approach used in first attempt to link Marc’s CL Prover (written in Prolog) to Isabelle
- Backend for producing Isabelle proof terms was derived from existing Coq backend

Problems

- Overhead for translating, parsing and printing
- Difficult to maintain: needs Prolog compiler to execute, must adapt Isabelle interface to changes of input or output format of external prover
- Scalability: proof terms might get too large
An internal prover

- Written in Isabelle’s implementation language (Standard ML)
- No parsing and printing of “external” formats
- Can work directly on Isabelle’s data structure for terms (and theorems)
- Uses existing infrastructure for
  - unification / matching
  - backtracking \(\leadsto\) sequences / lazy lists
  - managing large sets of facts \(\leadsto\) discrimination nets
Data structures

Rules

\[ \text{theorem} \rightarrow \text{types of } \exists\text{-quantified variables} \]
\[ \text{thm} \times \text{term list} \times (\text{typ list} \times \text{term list}) \text{ list} \]

conclusion

premises

conjuncts

Proofs

datatype cl_prf = ClPrf of

\[ \text{thm} \times \]
\[ (\text{Type.tyenv} \times \text{Envir.tenv}) \times \]
\[ ((\text{indexname} \times \text{typ}) \times \text{term}) \text{ list} \times \]
\[ \text{int list} \times \]
\[ (\text{term list} \times \text{cl_prf}) \text{ list} \]

theorem applied in proof step

instantiation for vars in premises of theorem

instantiation for extra vars

indices of facts used for solving premises

proofs for cases generated by theorem
The main loop

Construct the following (lazy) list:

For all rules

For all combinations of facts that make premises valid

For all instantiations of extra variables in conclusion

If conclusion is invalid, include (rule, facts, instantiation) in list

Otherwise do nothing

- order of rules matters (because of DFS strategy)
- try “oldest” facts first

Consider the first element of this list

- If there is no such element, we have found a countermodel
- If conclusion of chosen rule equals goal, we are done
- Otherwise recursively produce proofs of goal in all cases of conclusion of chosen rule
The main loop

fun valid0 thy rules goal dom facts nfacts nparams =
  let val seq = Seq.of_list rules |
    Seq.maps (fn (th, ps, cs) =>
      valid_conj thy facts empty_env ps |
    Seq.maps (fn (env, is) =>
      let val cs' = ⟨apply env to cs⟩
      in inst_extra_vars thy dom cs' |
        Seq.map_filter (fn (inst, cs''') =>
          if is_valid_disj thy facts cs''' then NONE
        else SOME (th, env, inst, is, cs'''))
    end)
  in case Seq.pull seq of
    NONE => NONE
    | SOME ((th, env, inst, is, cs), _) =>
      if cs = [([], [goal])] then SOME (ClPrf (th, env, inst, is, []))
      else (case valid2 thy rules goal dom facts nfacts nparams cs of
        NONE => NONE
        | SOME prfs => SOME (ClPrf (th, env, inst, is, prfs)))
    end
and valid2 thy rules goal dom facts nfacts nparams [] = SOME []
  | valid2 thy rules goal dom facts nfacts nparams ((Ts, ts) :: ds) =
    let
      val params = ⟨invent new parameters with types Ts⟩;
      val ts' = map_index (fn (i, t) =>
        (subst_bounds (params, t), nfacts + i)) ts;
      val dom' = ⟨add params to dom⟩;
      val facts' = ⟨add ts' to facts⟩
    in
    case valid0 thy rules goal dom' facts'
      (nfacts + length ts) (nparams + length Ts) of
      NONE => NONE
    | SOME prf =>
        (case valid2 thy rules goal dom facts facts' nparams ds of
          NONE => NONE
        | SOME prfs => SOME ((params, prf) :: prfs))
    end;
Proof Reconstruction

fun thm_of_cl_prf thy goal asms (ClPrf (th, env, insts, is, prfs)) = 
  let
    val th' = Drule.implies_elim_list 
    ∙⟨
      apply env and inst to th
    ⟩ (map (nth asms) is);
    val (_, cases) = dest_elim (prop_of th')
  in
    case (cases, prfs) of
      ([([], []), []]) => th'
    | ([([], []), [([], prf)]) =>
        thm_of_cl_prf thy goal (asms @ [th']) prf
    | _ => Drule.implies_elim_list 
        ∙⟨
          instantiate proposition var in th' with goal
        ⟩ (map (thm_of_case_prf thy goal asms) (prfs ~ cases))
  end
Proof Reconstruction – Case analysis

and thm_of_case_prf thy goal asms ((params, prf), (_, asms')) =
  let
    val cparams = map (cterm_of thy) params;
    val asms'' = map (cterm_of thy o curry subst_bounds (rev params)) asms'
  in
    Drule.forall_intr_list cparams (Drule.implies_intr_list asms''
      (thm_of_cl_prf thy goal (asms @ map Thm.assume asms'')) prf)
  end;
Conclusion
Future Work

• Use unification rather than enumeration of instantiations for extra variables in conclusion of a rule
  ↼ use ideas from free variable tableaux / hyper-tableaux [Furbach, Baumgartner]
• Extension to handling of function symbols
• Preprocessor / Translation from FOL to CL ↼ Andrew’s talk
• Different search strategies: BFS
Questions?