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

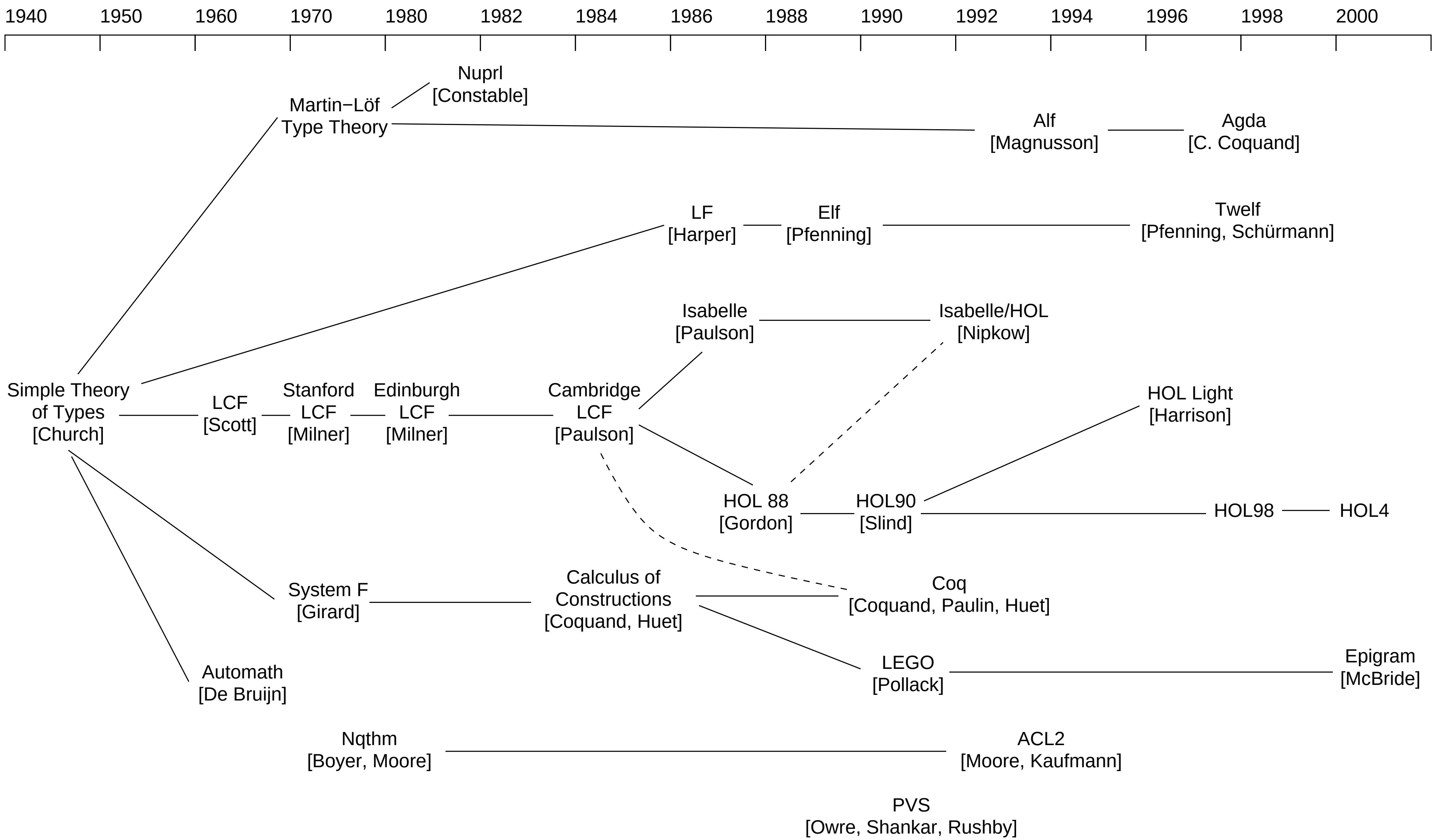
Isabelle

- 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.

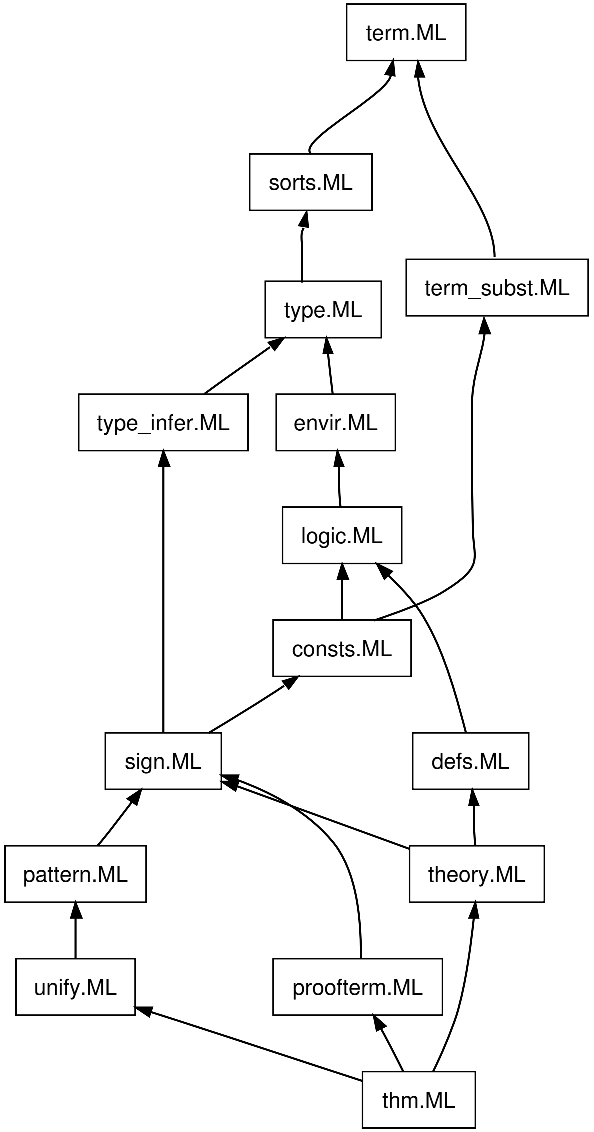
"The method of 'postulating' what we want has many advantages;
they are the same as the advantages of theft over honest toil.
Let us leave them to others and proceed with our honest toil."

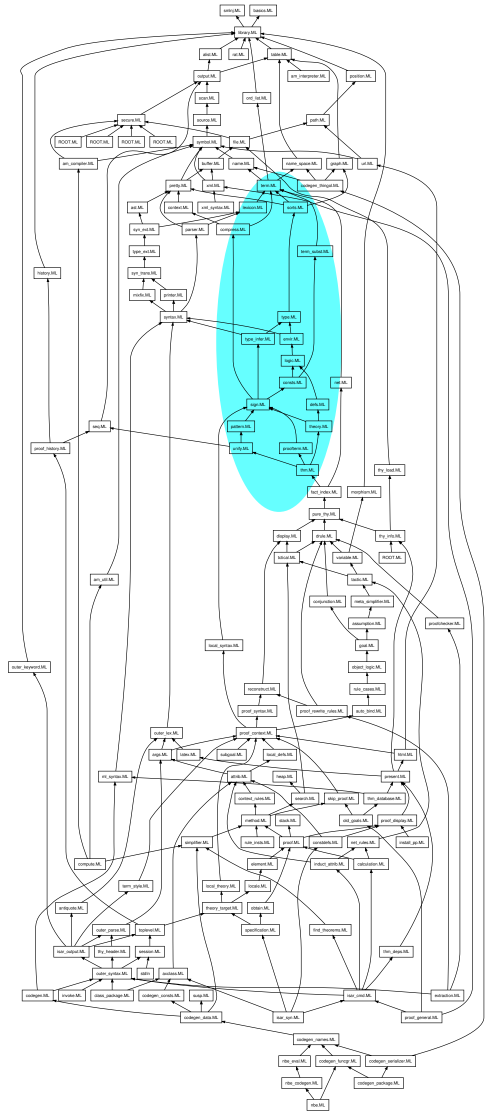
Bertrand Russell, Introduction to Mathematical Philosophy

A short history of theorem provers

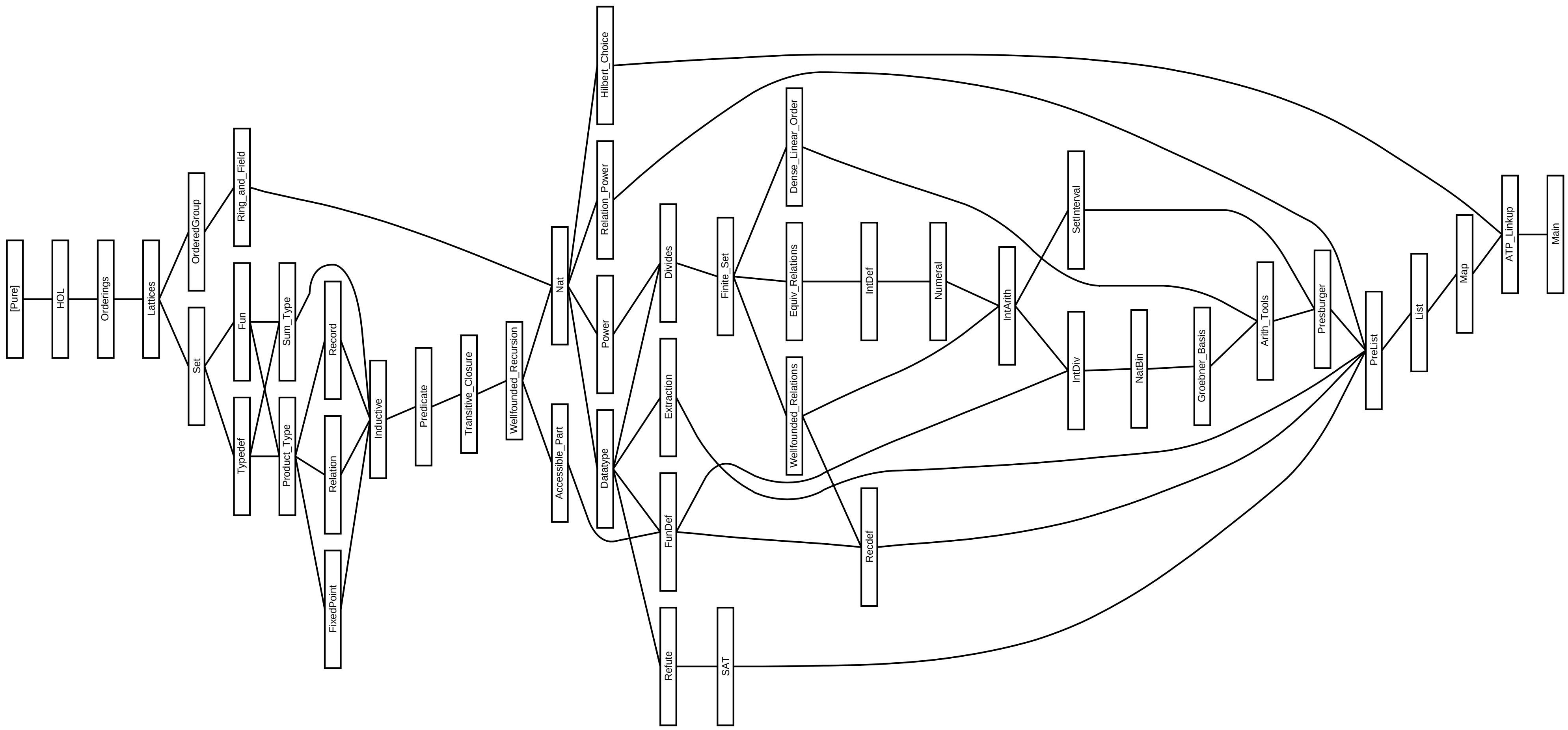


Architectue of Isabelle/Pure





Theory hierarchy of Isabelle/HOL



Isabelle's Logic

Formalizing logics in Isabelle

Meta logic Isabelle/Pure

- Terms: $t = x \mid c \mid \lambda x :: \tau.\ t \mid t\ t$
- Types: $\tau = \alpha \mid (\tau_1, \ldots, \tau_n) tc$ where $tc \in \{\Rightarrow,\ prop,\ \ldots\}$
- Logical operators:

Implication	$\Longrightarrow$	$::$	$prop \Rightarrow prop \Rightarrow prop$
Universal quantifier	$\bigwedge$	$::$	$(\alpha \Rightarrow prop) \Rightarrow prop$
Equality	$\equiv$	$::$	$\alpha \Rightarrow \alpha \Rightarrow prop$

Object logic Isabelle/HOL

- Terms and types: as in Isabelle/Pure
- Logical operators:

Truth predicate	$\lfloor \ldots \rfloor$	$::$	$bool \Rightarrow prop$
Conjunction	$\wedge$	$::$	$bool \Rightarrow bool \Rightarrow bool$
Disjunction	$\vee$	$::$	$bool \Rightarrow bool \Rightarrow bool$
Implication	$\longrightarrow$	$::$	$bool \Rightarrow bool \Rightarrow bool$
Universal quantifier	$\forall$	$::$	$(\alpha \Rightarrow bool) \Rightarrow bool$
Existential quantifier	$\exists$	$::$	$(\alpha \Rightarrow bool) \Rightarrow bool$

Proof representation in Isabelle/Pure

Proofs as λ-terms

$p, q \; = \; h$		Hypothesis
$\mid$	$c_{\{\overline{\alpha} \mapsto \overline{\tau}\}}$	Proof constant (reference to axiom / theorem)
$\mid$	$p \cdot t$	$\bigwedge$ -elimination
$\mid$	$p \bullet q$	$\Longrightarrow$ -elimination
$\mid$	$\boldsymbol{\lambda} x :: \tau.\ p$	$\bigwedge$ -introduction
$\mid$	$\boldsymbol{\lambda} h : \varphi.\ p$	$\Longrightarrow$ -introduction

Proof checking

$$\frac{}{\Gamma, h : t, \Gamma' \vdash h : t} \qquad \frac{\Sigma(c) = \varphi}{\Gamma \vdash c_{\{\overline{\alpha} \mapsto \overline{\tau}\}} : \varphi\{\overline{\alpha} \mapsto \overline{\tau}\}}$$

$$\frac{\Gamma \vdash p : \bigwedge x :: \tau.\ \varphi \quad \Gamma \vdash t :: \tau}{\Gamma \vdash p \cdot t : P\{x \mapsto t\}} \qquad \frac{\Gamma, x :: \tau \vdash p : \varphi}{\Gamma \vdash \boldsymbol{\lambda} x :: \tau.\ p : \bigwedge x :: \tau.\ \varphi}$$

$$\frac{\Gamma \vdash p : \varphi \Longrightarrow \psi \quad \Gamma \vdash q : \varphi}{\Gamma \vdash p \bullet q : \psi} \qquad \frac{\Gamma, h : \varphi \vdash p : \psi \quad \Gamma \vdash \varphi :: \mathsf{prop}}{\Gamma \vdash \boldsymbol{\lambda} h : \varphi.\ p : \varphi \Longrightarrow \psi}$$

Natural deduction calculus [Gentzen 1933]

Introduction rules

$$\frac{P \qquad Q}{P \wedge Q}\,(\wedge I)$$

$$\frac{P}{P \vee Q}\,(\vee I_1) \qquad\qquad \frac{Q}{P \vee Q}\,(\vee I_2)$$

$$\frac{\begin{matrix}[P]\\ \vdots \\ Q\end{matrix}}{P \longrightarrow Q}\,(\longrightarrow I)$$

Elimination rules

$$\frac{P \wedge Q \qquad \begin{matrix}[P,\ Q]\\ \vdots \\ R\end{matrix}}{R}\,(\wedge E)$$

$$\frac{P \vee Q \qquad \begin{matrix}[P]\\ \vdots \\ R\end{matrix} \qquad \begin{matrix}[Q]\\ \vdots \\ R\end{matrix}}{R}\,(\vee E)$$

$$\frac{P \longrightarrow Q \qquad P}{Q}\,(\longrightarrow E)$$

$$\frac{\bot}{P}\,(\bot E)$$

More rules

$$\frac{\begin{matrix}[P] \\ \vdots \\ \bot\end{matrix}}{\neg P}\,(\neg I) \qquad \frac{\neg P \quad P}{Q}\,(\neg E)$$

$$\frac{P}{\forall x.P}\,(\forall I)* \qquad \frac{\forall x.P}{P[t/x]}\,(\forall E)$$

$$\frac{P[t/x]}{\exists x.P}\,(\exists I) \qquad \frac{\exists x.P \quad \begin{matrix}[P] \\ \vdots \\ Q\end{matrix}}{Q}\,(\exists E)*$$

******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: $\lfloor P \rfloor \Longrightarrow \lfloor Q \rfloor \Longrightarrow \lfloor P \wedge Q \rfloor$

conjE: $\lfloor P \wedge Q \rfloor \Longrightarrow$
$(\lfloor P \rfloor \Longrightarrow \lfloor Q \rfloor \Longrightarrow \lfloor R \rfloor) \Longrightarrow \lfloor R \rfloor$

disjI1: $\lfloor P \rfloor \Longrightarrow \lfloor P \vee Q \rfloor$
disjI2: $\lfloor Q \rfloor \Longrightarrow \lfloor P \vee Q \rfloor$

disjE: $\lfloor P \vee Q \rfloor \Longrightarrow (\lfloor P \rfloor \Longrightarrow \lfloor R \rfloor) \Longrightarrow$
$(\lfloor Q \rfloor \Longrightarrow \lfloor R \rfloor) \Longrightarrow \lfloor R \rfloor$

impI: $(\lfloor P \rfloor \Longrightarrow \lfloor Q \rfloor) \Longrightarrow \lfloor P \longrightarrow Q \rfloor$

mp: $(\lfloor P \longrightarrow Q \rfloor) \Longrightarrow \lfloor P \rfloor \Longrightarrow \lfloor Q \rfloor$

FalseE: $\lfloor False \rfloor \Longrightarrow \lfloor P \rfloor$

notI: $(\lfloor P \rfloor \Longrightarrow \lfloor False \rfloor) \Longrightarrow \lfloor \neg P \rfloor$

notE: $\lfloor \neg P \rfloor \Longrightarrow \lfloor P \rfloor \Longrightarrow \lfloor Q \rfloor$

allI: $(\bigwedge x.\ \lfloor P\ x \rfloor) \Longrightarrow \lfloor \forall x.\ P\ x \rfloor$

spec: $\lfloor \forall x.\ P\ x \rfloor \Longrightarrow \lfloor P\ x \rfloor$

exI: $\lfloor P\ x \rfloor \Longrightarrow \lfloor \exists x.\ P\ x \rfloor$

exE: $\lfloor \exists x.\ P\ x \rfloor \Longrightarrow$
$(\bigwedge x.\ \lfloor P\ x \rfloor \Longrightarrow \lfloor Q \rfloor) \Longrightarrow \lfloor Q \rfloor$

Unstructured vs. structured proofs

theorem $ex1$: $(\exists\, x.\ \forall\, y.\ P\ x\ y) \longrightarrow (\forall\, y.\ \exists\, x.\ P\ x\ y)$
  apply $(rule\ impI)$
  apply $(erule\ exE)$
  apply $(rule\ allI)$
  apply $(rule\ exI)$
  apply $(drule\ spec)$
  apply $assumption$
  done

theorem $ex2$: $(\exists\, x.\ \forall\, y.\ P\ x\ y) \longrightarrow (\forall\, y.\ \exists\, x.\ P\ x\ y)$
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

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?

$$A_1 \wedge \ldots \wedge A_m \longrightarrow (\exists \vec{x_1}.\ B_1^1 \wedge \ldots \wedge B_1^{k_1}) \vee \ldots \vee (\exists \vec{x_n}.\ B_n^1 \wedge \ldots \wedge B_n^{k_n})$$

$$\equiv$$

$$A_1 \Longrightarrow \cdots \Longrightarrow A_n \Longrightarrow (\bigwedge \vec{x_1}.\ B_1^1 \Longrightarrow \cdots \Longrightarrow B_1^{k_1} \Longrightarrow P) \Longrightarrow \cdots$$
$$\Longrightarrow (\bigwedge \vec{x_n}.\ B_n^1 \Longrightarrow \cdots \Longrightarrow B_n^{k_n} \Longrightarrow P) \Longrightarrow P$$

Rules used in the translation

$$\begin{aligned}
A &\equiv (\bigwedge B.\ (A \Longrightarrow B) \Longrightarrow B) \\
(A \wedge B \Longrightarrow C) &\equiv (A \Longrightarrow B \Longrightarrow C) \\
((A \vee B \Longrightarrow C) \Longrightarrow C) &\equiv ((A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C) \\
((\exists x.\ P\ x) \Longrightarrow Q) &\equiv (\bigwedge x.\ P\ x \Longrightarrow Q)
\end{aligned}$$

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 $\rightsquigarrow$ sequences / lazy lists
  - managing large sets of facts $\rightsquigarrow$ discrimination nets

Data structures

Rules

theorem: `thm`; types of ∃-quantified variables: `typ list`; premises: `term list`; conjuncts: `term list`; conclusion: `(typ list * term list) list`

```
thm * term list * (typ list * term list) list
```

Proofs

```
datatype cl_prf = ClPrf of
  thm *                               theorem applied in proof step
  (Type.tyenv * Envir.tenv) *         instantiation for vars in premises of theorem
  ((indexname * typ) * term) list *   instantiation for extra vars
  int list *                          indices of facts used for solving premises
  (term list * cl_prf) list           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 conlusion
      - **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
```

Case analysis

```
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 nfacts 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
  $\rightsquigarrow$ use ideas from free variable tableaux / hyper-tableaux [Furbach, Baumgartner]
- Extension to handling of function symbols
- Preprocessor / Translation from FOL to CL $\rightsquigarrow$ Andrew's talk
- Different search strategies: BFS

Questions?