The Isabelle Prover IDE (PIDE) after 9 years of development, and beyond

Makarius Wenzel
http://sketis.net

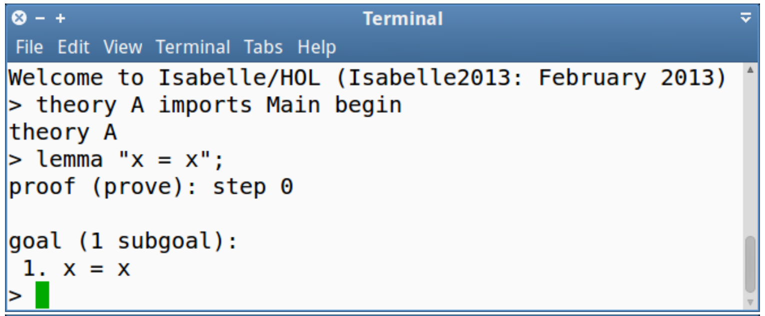
July 2017

Abstract

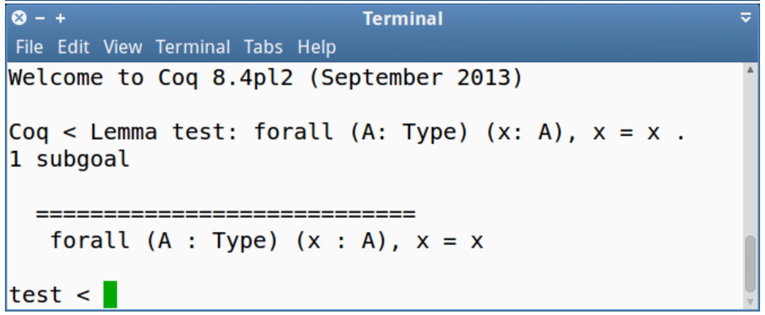
The main ideas around Isabelle/PIDE go back to summer 2008. This is an overview of what has been achieved in the past 9 years, with some prospects for the future. Where can we go from here as Isabelle community? (E.g. towards alternative front-ends like Visual Studio Code; remote prover sessions "in the cloud"; support for collaborative editing of large formal libraries.) Where can we go as greater ITP community (Lean, Coq, HOL family)?

History of Prover Interaction

TTY loop ($\approx$ 1979)



```
Welcome to Isabelle/HOL (Isabelle2013: February 2013)
> theory A imports Main begin
theory A
> lemma "x = x";
proof (prove): step 0

goal (1 subgoal):
 1. x = x
> 
```

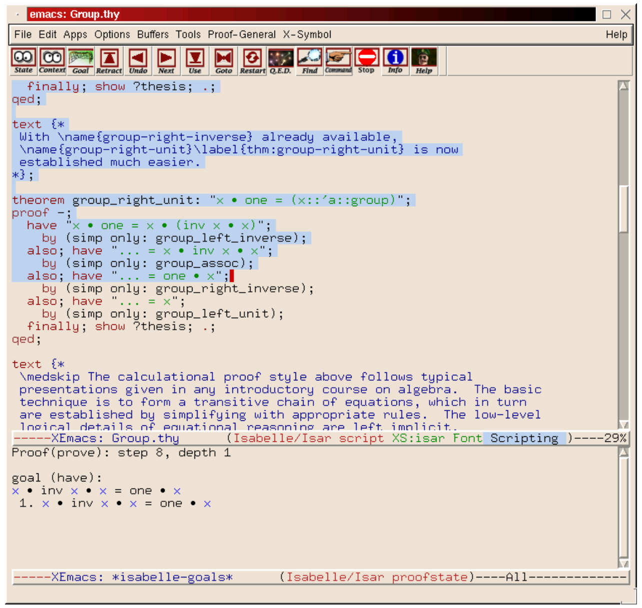


```
Welcome to Coq 8.4pl2 (September 2013)

Coq < Lemma test: forall (A: Type) (x: A), x = x .
1 subgoal

  ============================
   forall (A : Type) (x : A), x = x

test < 
```

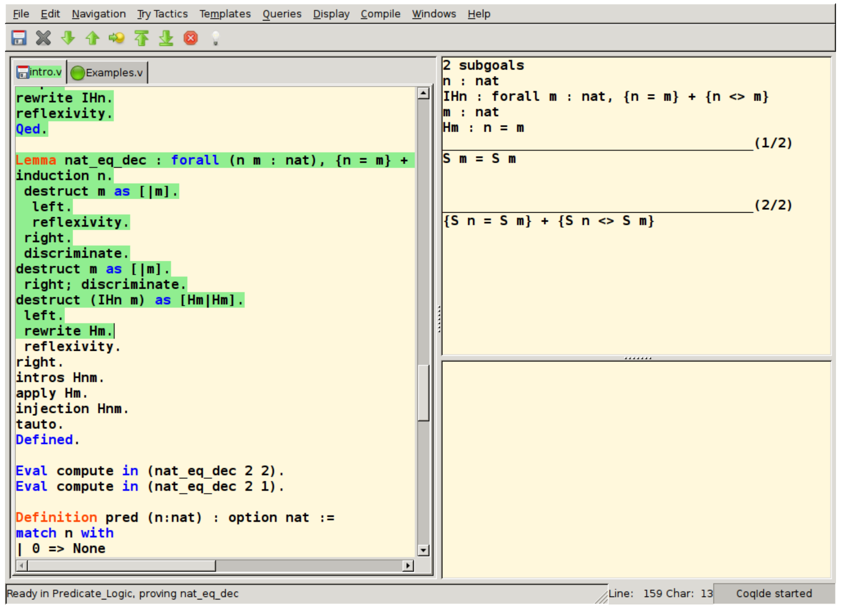
(Wikipedia: K. Thompson and D. Ritchie at PDP-11)

- user drives prover, via manual copy-paste
- synchronous and sequential

Proof General and clones (≈ 1999)


emacs: Group.thy
File Edit Apps Options Buffers Tools Proof-General X-Symbol Help
State Context Goal Retract Undo Next Use Goto Restart Q.E.D. Find Command Stop Info Help

```
  finally; show ?thesis; .;
qed;

text {*
 With \name{group-right-inverse} already available,
 \name{group-right-unit}\label{thm:group-right-unit} is now
 established much easier.
*};

theorem group_right_unit: "x • one = (x::'a::group)";
proof -;
  have "x • one = x • (inv x • x)";
    by (simp only: group_left_inverse);
  also; have "... = x • inv x • x";
    by (simp only: group_assoc);
  also; have "... = one • x";
    by (simp only: group_right_inverse);
  also; have "... = x";
    by (simp only: group_left_unit);
  finally; show ?thesis; .;
qed;

text {*
 \medskip The calculational proof style above follows typical
 presentations given in any introductory course on algebra.  The basic
 technique is to form a transitive chain of equations, which in turn
 are established by simplifying with appropriate rules.  The low-level
 logical details of equational reasoning are left implicit.
-----XEmacs: Group.thy       (Isabelle/Isar script XS:isar Font Scripting )----29%
Proof(prove): step 8, depth 1

goal (have):
x • inv x • x = one • x
 1. x • inv x • x = one • x

-----XEmacs: *isabelle-goals*       (Isabelle/Isar proofstate)----All-------------
```


File Edit Navigation Try Tactics Templates Queries Display Compile Windows Help
intro.v
Examples.v

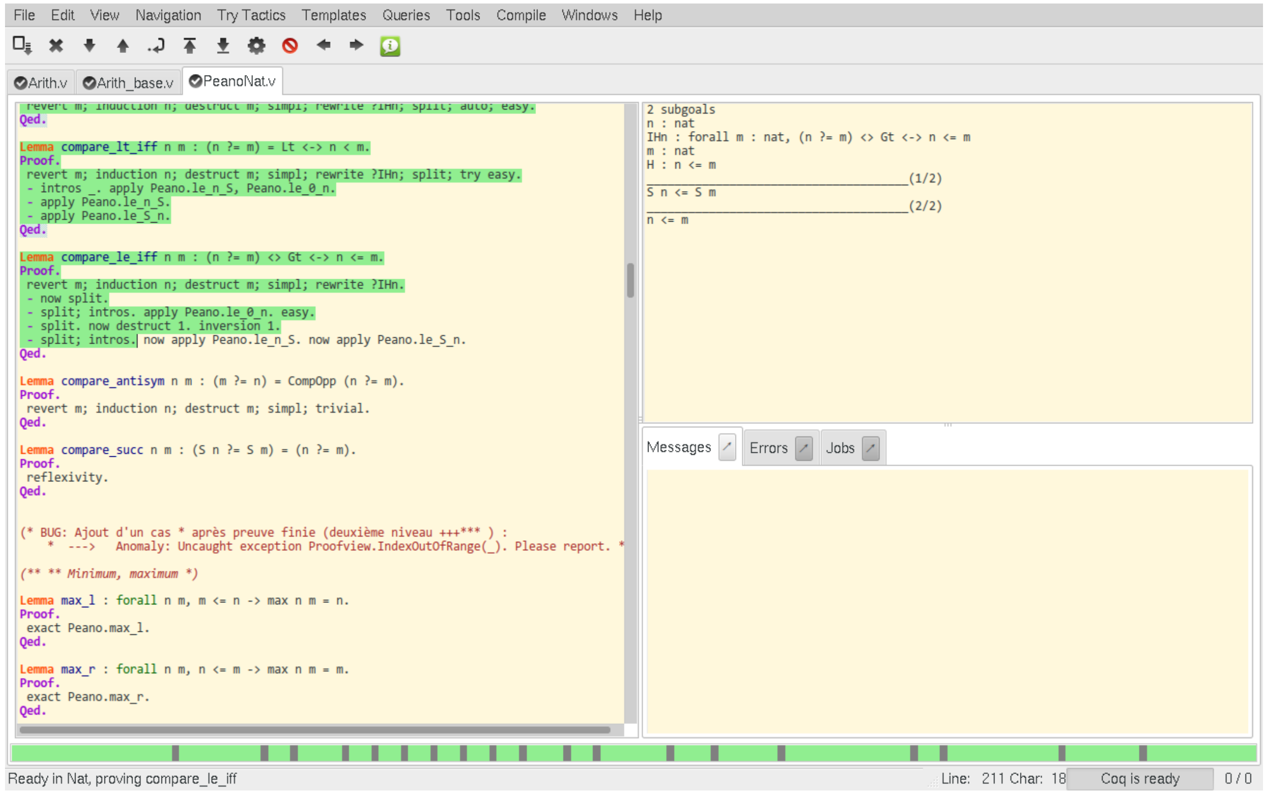
```
rewrite IHn.
reflexivity.
Qed.

Lemma nat_eq_dec : forall (n m : nat), {n = m} +
induction n.
 destruct m as [|m].
  left.
  reflexivity.
 right.
 discriminate.
destruct m as [|m].
 right; discriminate.
destruct (IHn m) as [Hm|Hm].
 left.
 rewrite Hm.
 reflexivity.
right.
intros Hnm.
apply Hm.
injection Hnm.
tauto.
Defined.

Eval compute in (nat_eq_dec 2 2).
Eval compute in (nat_eq_dec 2 1).

Definition pred (n:nat) : option nat :=
match n with
| 0 => None
```

```
2 subgoals
n : nat
IHn : forall m : nat, {n = m} + {n <> m}
m : nat
Hm : n = m
______________________________________(1/2)
S m = S m
______________________________________(2/2)
{S n = S m} + {S n <> S m}
```

```
Ready in Predicate_Logic, proving nat_eq_dec        Line:  159 Char:  13    CoqIde started
```

- user drives prover, via automated copy-paste and undo
- synchronous and sequential

CoqIDE (≈ 2016)


File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help
Arith.v
Arith_base.v
PeanoNat.v
Messages
Errors
Jobs
Coq is ready
0 / 0

```
 revert m; induction n; destruct m; simpl; rewrite ?IHn; split; auto; easy.
Qed.

Lemma compare_lt_iff n m : (n ?= m) = Lt <-> n < m.
Proof.
 revert m; induction n; destruct m; simpl; rewrite ?IHn; split; try easy.
 - intros _. apply Peano.le_n_S, Peano.le_0_n.
 - apply Peano.le_n_S.
 - apply Peano.le_S_n.
Qed.

Lemma compare_le_iff n m : (n ?= m) <> Gt <-> n <= m.
Proof.
 revert m; induction n; destruct m; simpl; rewrite ?IHn.
 - now split.
 - split; intros. apply Peano.le_0_n. easy.
 - split. now destruct 1. inversion 1.
 - split; intros. now apply Peano.le_n_S. now apply Peano.le_S_n.
Qed.

Lemma compare_antisym n m : (m ?= n) = CompOpp (n ?= m).
Proof.
 revert m; induction n; destruct m; simpl; trivial.
Qed.

Lemma compare_succ n m : (S n ?= S m) = (n ?= m).
Proof.
 reflexivity.
Qed.


(* BUG: Ajout d'un cas * après preuve finie (deuxième niveau +++*** ) :
    *  --->   Anomaly: Uncaught exception Proofview.IndexOutOfRange(_). Please report. *

(** ** Minimum, maximum *)

Lemma max_l : forall n m, m <= n -> max n m = n.
Proof.
 exact Peano.max_l.
Qed.

Lemma max_r : forall n m, n <= m -> max n m = m.
Proof.
 exact Peano.max_r.
Qed.
```

```
2 subgoals
n : nat
IHn : forall m : nat, (n ?= m) <> Gt <-> n <= m
m : nat
H : n <= m
______________________________________(1/2)
S n <= S m
______________________________________(2/2)
n <= m
```

```
Ready in Nat, proving compare_le_iff          Line:  211 Char:  18
```

- more formal interaction protocol
- recent support for asynchronous proofs

PIDE: Prover IDE ($\approx$ 2008)

Approach:

Prover supports asynchronous document model natively

Editor continuously sends source edits and receives markup reports

Tools may participate in document processing and markup

User constructs document content — assisted by
GUI rendering of cumulative PIDE markup

PIDE: Prover IDE ($\approx$ 2008)

Approach:

Prover supports asynchronous document model natively

Editor continuously sends source edits and receives markup reports

Tools may participate in document processing and markup

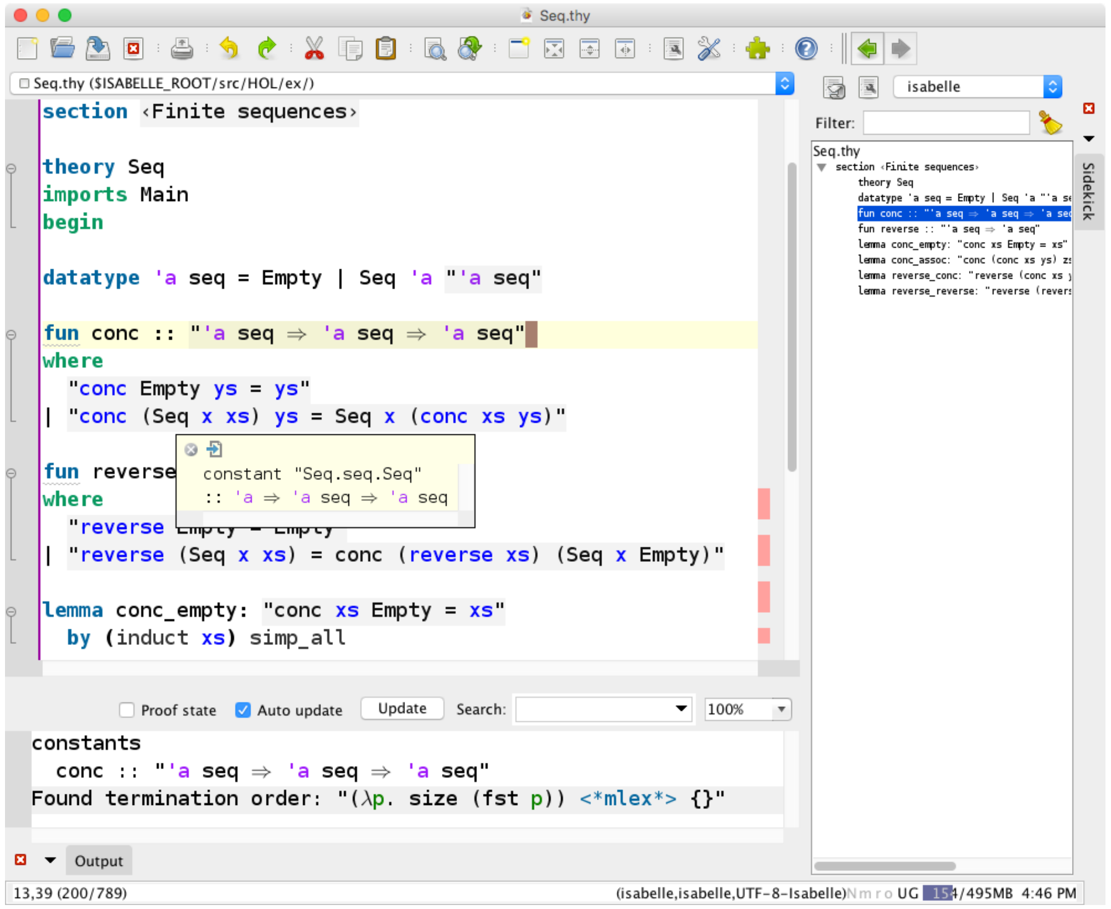
User constructs document content — assisted by
GUI rendering of cumulative PIDE markup

Challenge: introducing genuine interaction into ITP

- many conceptual problems
- many technical problems
- many social problems

Isabelle/jEdit Prover IDE (≈ 2016)

- asynchronous interaction
- continuous checking
- parallel processing


Seq.thy
Seq.thy ($ISABELLE_ROOT/src/HOL/ex/)
isabelle
Filter:
Sidekick
Proof state
Auto update
Update
Search:
100%
Output
13,39 (200/789)
(isabelle,isabelle,UTF-8-Isabelle)Nmro UG 154/495MB 4:46 PM

```
section ‹Finite sequences›

theory Seq
imports Main
begin

datatype 'a seq = Empty | Seq 'a "'a seq"

fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

fun reverse
where
  "reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"

lemma conc_empty: "conc xs Empty = xs"
  by (induct xs) simp_all
```

```
constant "Seq.seq.Seq"
:: 'a ⇒ 'a seq ⇒ 'a seq
```

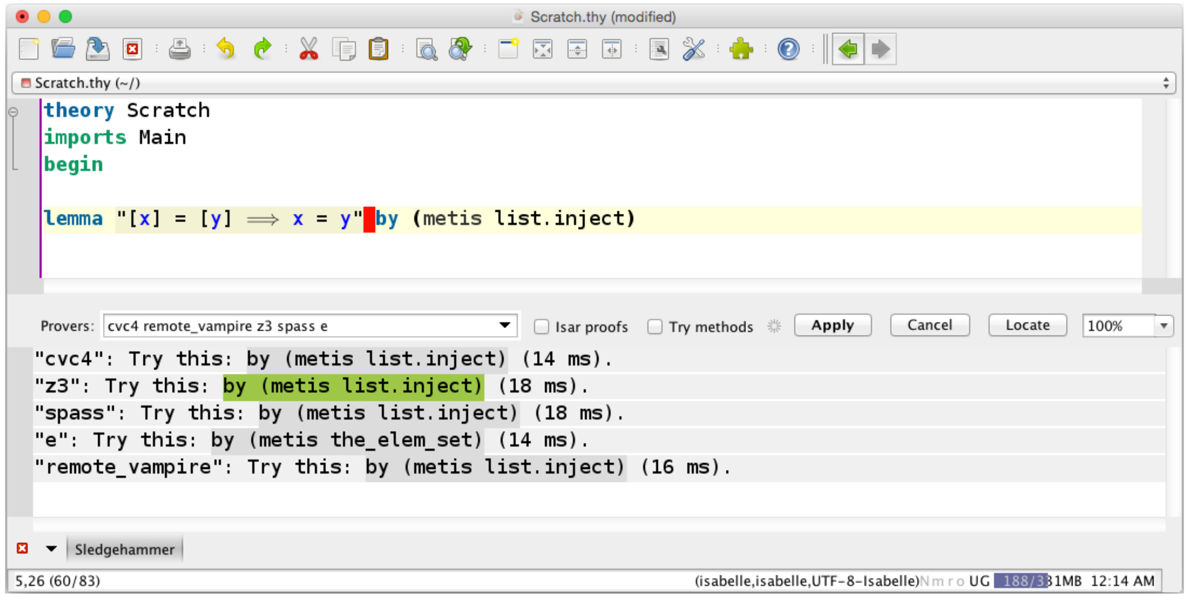
```
Seq.thy
▼ section ‹Finite sequences›
    theory Seq
    datatype 'a seq = Empty | Seq 'a "'a se
    fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq
    fun reverse :: "'a seq ⇒ 'a seq"
    lemma conc_empty: "conc xs Empty = xs"
    lemma conc_assoc: "conc (conc xs ys) zs
    lemma reverse_conc: "reverse (conc xs y
    lemma reverse_reverse: "reverse (revers
```

```
constants
  conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
```

Isabelle/jEdit: tool integration

Sledgehammer:

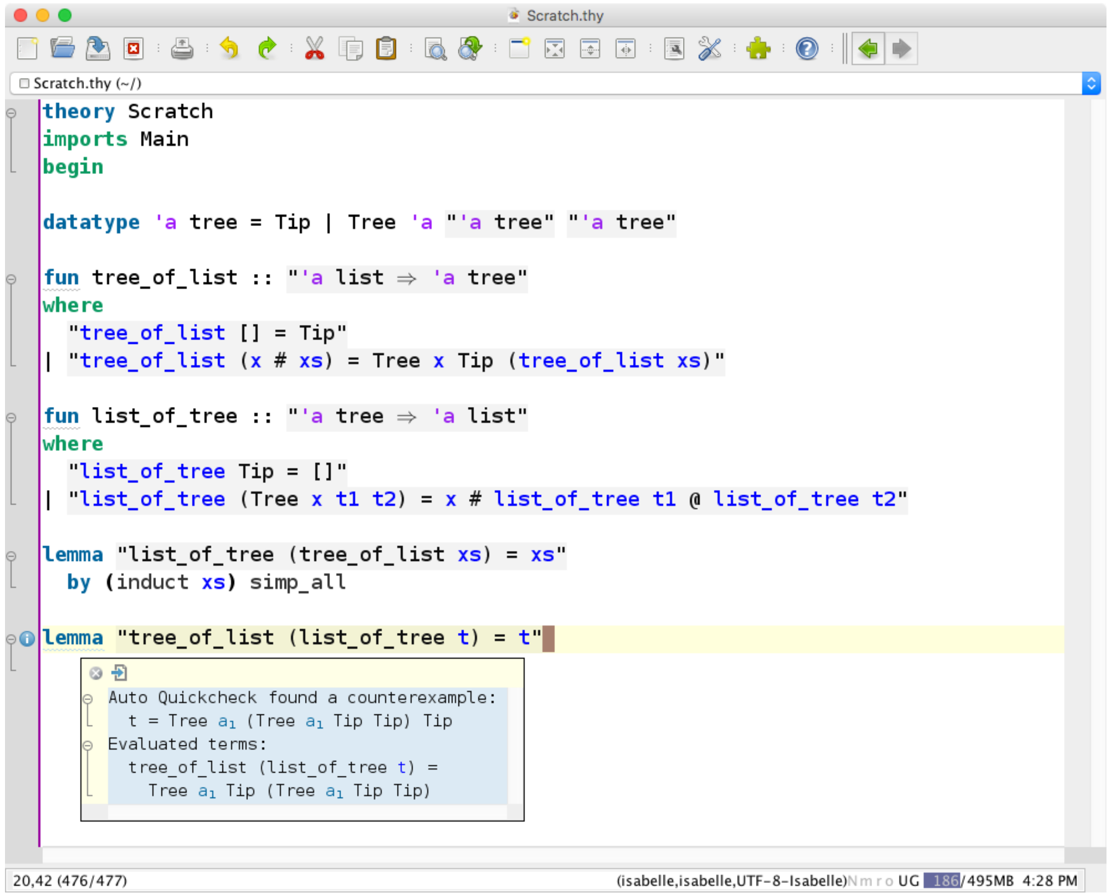
- integration with automated reasoning tools
- heavy external ATPs / SMTs for proof search
- light internal ATP (Metis) for proof reconstruction



```
theory Scratch
imports Main
begin

lemma "[x] = [y] ⟹ x = y" by (metis list.inject)
```

```
"cvc4": Try this: by (metis list.inject) (14 ms).
"z3": Try this: by (metis list.inject) (18 ms).
"spass": Try this: by (metis list.inject) (18 ms).
"e": Try this: by (metis the_elem_set) (14 ms).
"remote_vampire": Try this: by (metis list.inject) (16 ms).
```

Isabelle/jEdit: automatically tried tools



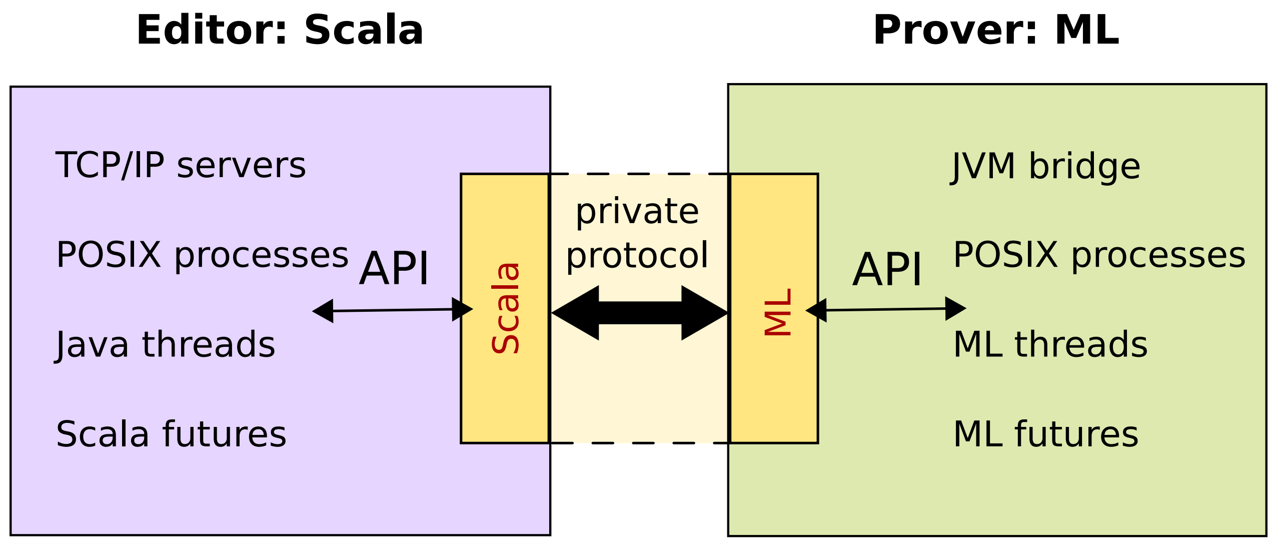
```
theory Scratch
imports Main
begin

datatype 'a tree = Tip | Tree 'a "'a tree" "'a tree"

fun tree_of_list :: "'a list ⇒ 'a tree"
where
  "tree_of_list [] = Tip"
| "tree_of_list (x # xs) = Tree x Tip (tree_of_list xs)"

fun list_of_tree :: "'a tree ⇒ 'a list"
where
  "list_of_tree Tip = []"
| "list_of_tree (Tree x t1 t2) = x # list_of_tree t1 @ list_of_tree t2"

lemma "list_of_tree (tree_of_list xs) = xs"
  by (induct xs) simp_all

lemma "tree_of_list (list_of_tree t) = t"
```

```
Auto Quickcheck found a counterexample:
  t = Tree a₁ (Tree a₁ Tip Tip) Tip
Evaluated terms:
  tree_of_list (list_of_tree t) =
    Tree a₁ Tip (Tree a₁ Tip Tip)
```

```
20,42 (476/477)          (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 186/495MB  4:28 PM
```

PIDE architecture

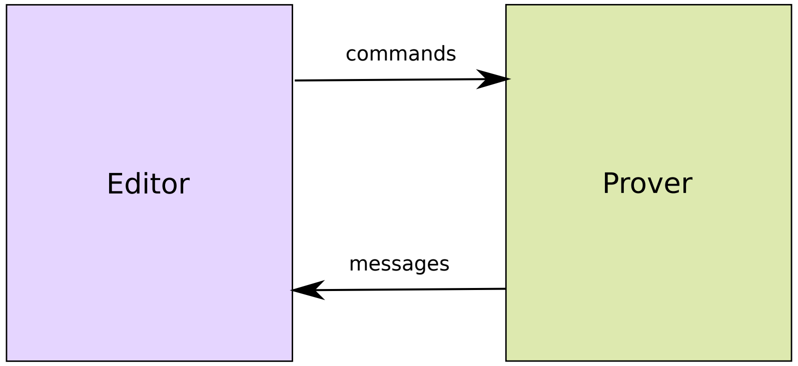
The connectivity problem



Design principles:

- private protocol for prover connectivity
  (asynchronous interaction, parallel evaluation)
- public Scala API
  (timeless, stateless, static typing)

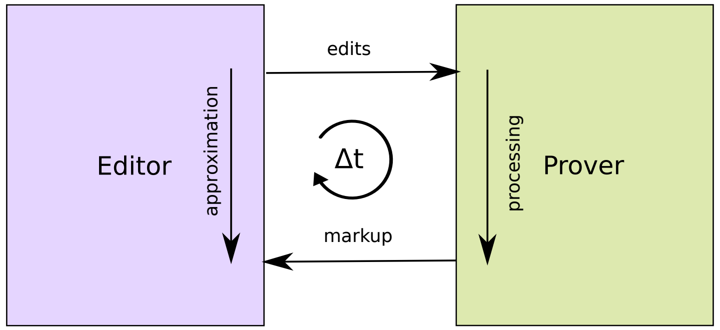
PIDE protocol functions



- $type\ protocol\_command = name \to input \to unit$
- $type\ protocol\_message = name \to output \to unit$
- outermost state of protocol handlers on each side (pure values)
- asynchronous streaming in each direction

$\longrightarrow$ editor and prover as stream-procession functions

Approximative rendering of document snapshots



1. editor knows text T, markup M, and edits ΔT (produced by user)
2. apply edits: $T' = T + \Delta T$ (immediately in editor)
3. formal processing of T': ΔM after time Δt (eventually in prover)
4. temporary approximation (immediately in editor):
   $\tilde{M} = \mathit{revert}\ \Delta T; \mathit{retrieve}\ M; \mathit{convert}\ \Delta T$
5. convergence after time Δt (eventually in editor):
   $M' = M + \Delta M$

Isabelle/jEdit

Building blocks

jEdit: http://www.jedit.org

- sophisticated text editor implemented in Java

Scala: http://www.scala-lang.org

- higher-order functional-object-oriented programming on JVM

PIDE:

- general framework for Prover IDEs based on Scala
- with parallel and asynchronous document processing

Isabelle/jEdit:

- main example application of the PIDE framework
- default user-interface for Isabelle
- filthy rich client: requires 4–8 GB memory, 2–4 CPU cores

Timeline

Parallel Isabelle

- 2005 “free lunch is over”: multicore invasion into consumer market
- 2006–2008 Isabelle + Poly/ML support multicore hardware in batch mode

Isabelle/PIDE/jEdit

- 2008–2010: experimental Isabelle/jEdit Prover IDE
- October 2011: stable release of Isabelle/jEdit 1.0
- December 2016: Isabelle/jEdit 8.0
- October 2017 (?): Isabelle/jEdit 9.0

Isabelle/VSCode

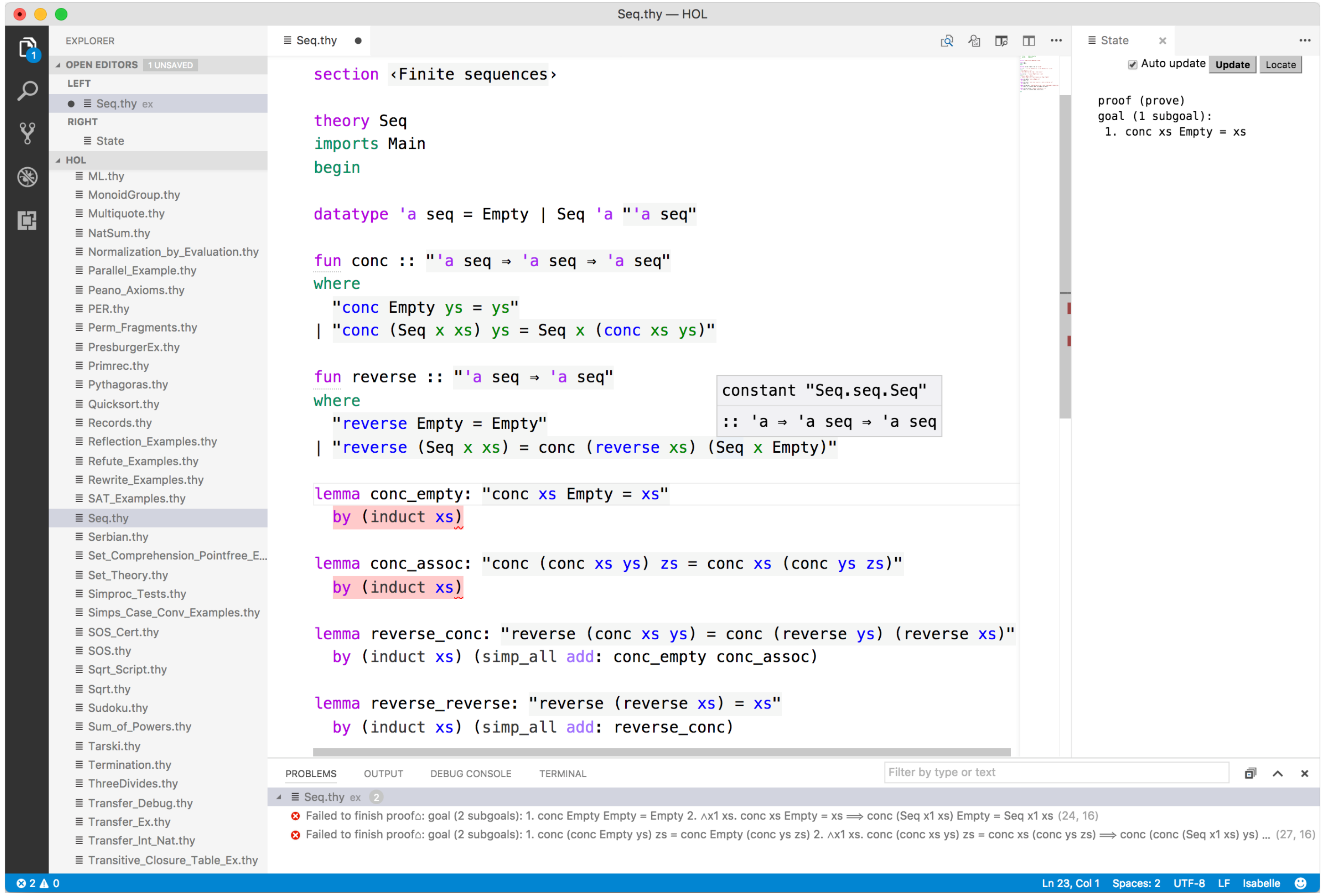
- Early 2017: experimental Isabelle/VSCode Prover IDE
- October 2017 (?): experimental Isabelle/VSCode 1.0.0

Isabelle/VSCode

Building blocks

- VSCode editor platform:
  - recent open-source project by Microsoft
    "Code editing. Redefined. Free. Open Source. Runs everywhere."
  - based on Electron application framework
    with Node.js, Chromium browser, V8 JavaScript engine
  - IDE for TypeScript in TypeScript (typed JavaScript)
- Isabelle/Scala/PIDE: slightly reworked for multiple front-ends
- Isabelle extension: via VSCode marketplace

Isabelle/VSCode: document-oriented interaction



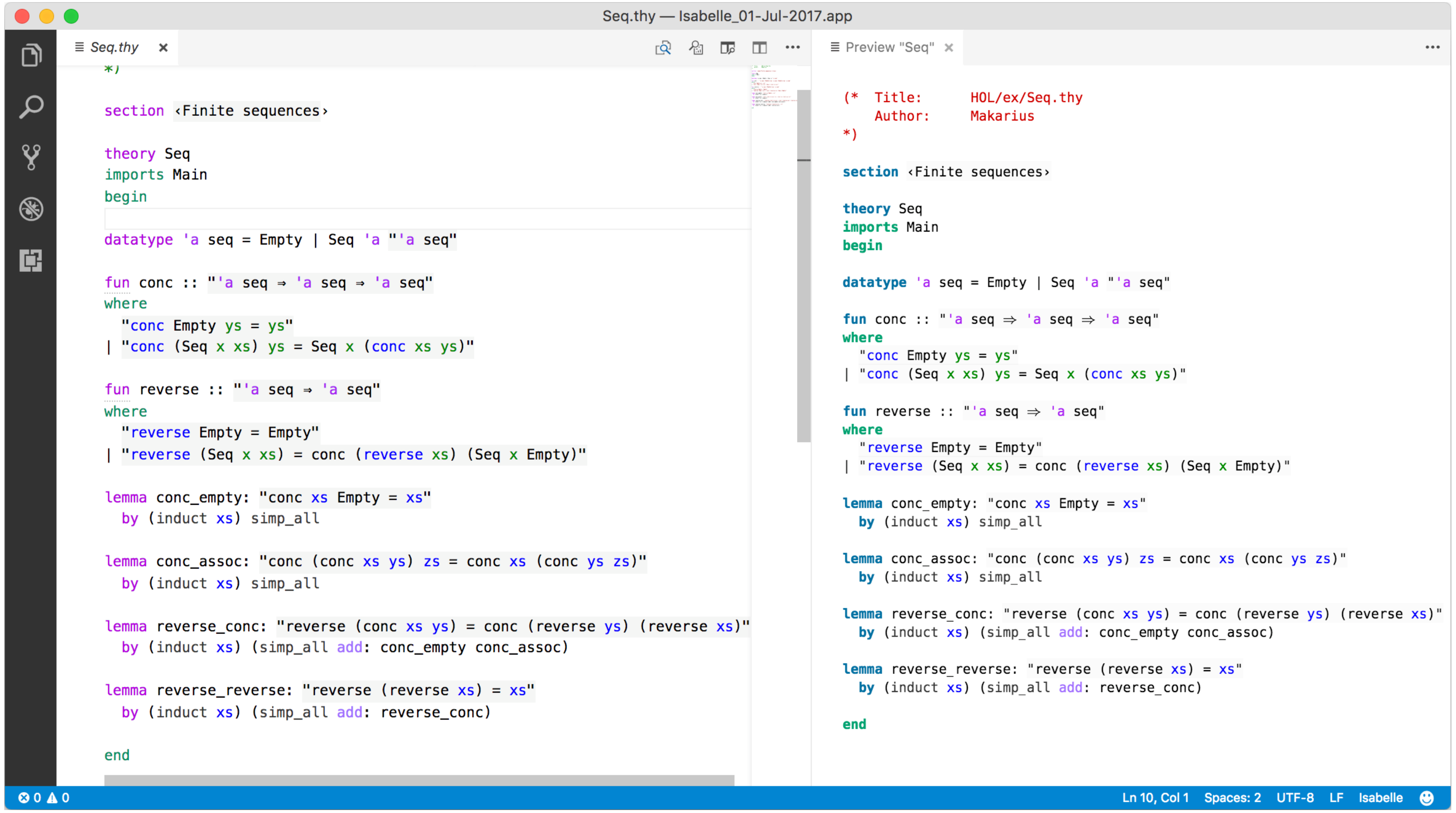
```
section ‹Finite sequences›

theory Seq
imports Main
begin

datatype 'a seq = Empty | Seq 'a "'a seq"

fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

fun reverse :: "'a seq ⇒ 'a seq"
where
  "reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"

lemma conc_empty: "conc xs Empty = xs"
  by (induct xs)

lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
  by (induct xs)

lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
  by (induct xs) (simp_all add: conc_empty conc_assoc)

lemma reverse_reverse: "reverse (reverse xs) = xs"
  by (induct xs) (simp_all add: reverse_conc)
```

```
constant "Seq.seq.Seq"
:: 'a ⇒ 'a seq ⇒ 'a seq
```

```
proof (prove)
goal (1 subgoal):
 1. conc xs Empty = xs
```

```
Failed to finish proof⌂: goal (2 subgoals): 1. conc Empty Empty = Empty 2. ⋀x1 xs. conc xs Empty = xs ⟹ conc (Seq x1 xs) Empty = Seq x1 xs (24, 16)
Failed to finish proof⌂: goal (2 subgoals): 1. conc (conc Empty ys) zs = conc Empty (conc ys zs) 2. ⋀x1 xs. conc (conc xs ys) zs = conc xs (conc ys zs) ⟹ conc (conc (Seq x1 xs) ys) ... (27, 16)
```

Isabelle/VSCode: document preview



Seq.thy — Isabelle_01-Jul-2017.app

Seq.thy

```
*)

section ‹Finite sequences›

theory Seq
imports Main
begin

datatype 'a seq = Empty | Seq 'a "'a seq"

fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

fun reverse :: "'a seq ⇒ 'a seq"
where
  "reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"

lemma conc_empty: "conc xs Empty = xs"
  by (induct xs) simp_all

lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
  by (induct xs) simp_all

lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
  by (induct xs) (simp_all add: conc_empty conc_assoc)

lemma reverse_reverse: "reverse (reverse xs) = xs"
  by (induct xs) (simp_all add: reverse_conc)

end
```

Preview "Seq"

```
(*  Title:      HOL/ex/Seq.thy
    Author:     Makarius
*)

section ‹Finite sequences›

theory Seq
imports Main
begin

datatype 'a seq = Empty | Seq 'a "'a seq"

fun conc :: "'a seq ⇒ 'a seq ⇒ 'a seq"
where
  "conc Empty ys = ys"
| "conc (Seq x xs) ys = Seq x (conc xs ys)"

fun reverse :: "'a seq ⇒ 'a seq"
where
  "reverse Empty = Empty"
| "reverse (Seq x xs) = conc (reverse xs) (Seq x Empty)"

lemma conc_empty: "conc xs Empty = xs"
  by (induct xs) simp_all

lemma conc_assoc: "conc (conc xs ys) zs = conc xs (conc ys zs)"
  by (induct xs) simp_all

lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (reverse xs)"
  by (induct xs) (simp_all add: conc_empty conc_assoc)

lemma reverse_reverse: "reverse (reverse xs) = xs"
  by (induct xs) (simp_all add: reverse_conc)

end
```

Other VSCode prover projects

VSCoq: Coq Support for Visual Studio Code

- by C. J. Bell (MIT), see https://github.com/siegebell/vscoq
- uses Asynchronous Proofs from Coq/Paral-ITP project
- uses Coq XML protocol
- provides some HTML GUI components

Lean for VSCode

- by J. Roesch (Univ. Washington) and others,
  see https://github.com/leanprover/vscode-lean
- uses Lean server for incremental compilation and checking
- already used in practice

Potential of the VSCode/Electron platform

Technology

- High-quality HTML rendering within just one browser
- Perspective for viable multiplatform support (beyond Linux/X11)
- Generic GUI integration for:
  - Debugger
  - Version Control System (Git, Mercurial, . . .)

Ecosystem

- Project managed by developers at Microsoft
- Young and active community
- Many emerging projects and extensions

Future Work

Future Work

Scaling

- editing big libraries as a whole,
  notably The Archive of Formal Proofs
- offline PIDE markup in database files (e.g. SQLite)
- online PIDE markup in database server (e.g. PostgreSQL)
- integration with Version Control (e.g. Mercurial within VSCode)

Publishing

- backend: headless PIDE for "cloud" (e.g. via SSH or WebSocket)
- frontend: high-quality HTML presentation (e.g. via VSCode)
- advanced of formal publishing: LaTeX and HTML / CSS / MathJax
- PIDE as webserver / cloud service?