

Isabelle/jEdit for seasoned Isabelle users

Isabelle/jEdit NEWS

Makarius Wenzel
Univ. Paris-Sud, LRI

13-Jul-2014

“There is nothing to prove or even to suppose that the Serbian government is accessory to the inducement for the crime, its preparations, or the furnishing of weapons. On the contrary, there are reasons to believe that this altogether out of the question.”

(The July Crisis, 13-Jul-1914)

David Fromkin, 2004: *Europe's Last Summer: Who started the Great War in 1914?*

History

2009 (experimental)

```
File Edit Search Markers Folding View Utilities Macros Plugins Help
Test.thy (~tmp/)
theory Test
imports Main
begin

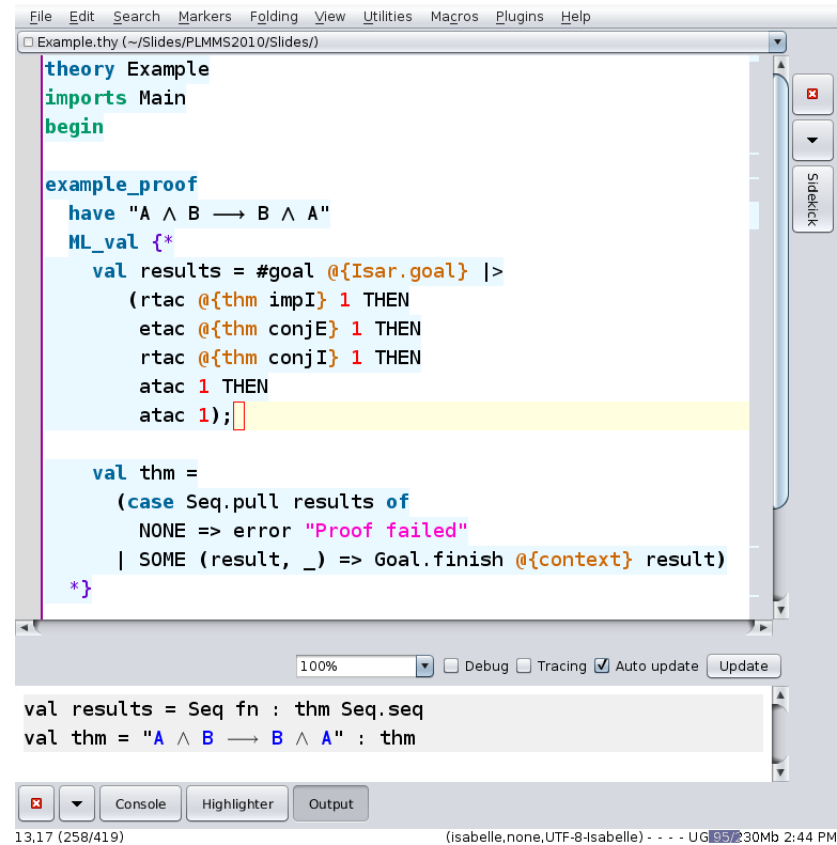
inductive path for rel :: "'a => 'a => bool" where
  base: "path rel x x"
| step: "rel x y => path rel y z => path rel x z"

theorem example:
  fixes x z :: 'a assumes path: "path rel x z" shows "P x z"
  using path
proof induct
  case (base x)
  show "P x x" sorry
next
  case (step x y z)
  note `rel x y` and `path rel y z`
  moreover note `P y z`
  ultimately show "P x z" sorry
qed

ML {* Thm.prop-of @ {thm example} *}
      Thm.prop_of: thm -> term
end

22.7 (422/456) (isabelle.none.UTF-8-Isabelle) - - - UE 1.2/1.4Mb 5:21 PM
```

2010 (experimental)



The screenshot shows a theorem prover interface with a menu bar (File, Edit, Search, Markers, Folding, View, Utilities, Macros, Plugins, Help) and a window titled 'Example.thy (~/Slides/PLMMS2010/Slides/)'. The main editor contains the following code:

```
theory Example
imports Main
begin

example_proof
  have "A & B -> B & A"
  ML_val {*
    val results = #goal @{Isar.goal} |>
      (rtac @{thm impI} 1 THEN
       etac @{thm conjE} 1 THEN
       rtac @{thm conjI} 1 THEN
       atac 1 THEN
       atac 1);
    val thm =
      (case Seq.pull results of
       NONE => error "Proof failed"
      | SOME (result, _) => Goal.finish @{context} result)
  *}

val results = Seq fn : thm Seq.seq
val thm = "A & B -> B & A" : thm
```

Below the editor is a status bar with a 100% zoom level, checkboxes for Debug, Tracing, and Auto update (checked), and an Update button. At the bottom, there are buttons for Console, Highlighter, and Output. The status bar also shows the text '(isabelle:none,UTF-8-Isabelle) - - - UG 35 30Mb 2:44 PM' and '13.17 (258/419)'.

2011 (experimental \rightsquigarrow stable)

The screenshot shows the Isabelle IDE interface. The main editor displays a proof script for a theorem named 'Example'. The script is as follows:

```
theory Example imports Main
begin

notepad
begin
  have "A  $\wedge$  B  $\longrightarrow$  B  $\wedge$  A"
  ML_val {
    val results = #goal @{Isar.goal} |>
      (rtac @{thm impI} 1 THEN
       etac @{thm conjE} 1 THEN
       rtac @{thm conjI} 1 THEN
       atac 1 THEN atac 1);
    val thm =
      (case Seq.pull results of
       NONE => error "Proof failed"
      | SOME (result, _) => Goal.finish @{context} result 1);
  }
 oops
end
```

The 'notepad' section is highlighted in yellow. A type error message is displayed in a yellow box at the bottom right of the editor:

```
Type error in function application.
Function:
  Goal.finish Isabelle.context
  result
  : thm
Argument: 1 : int
Reason:
  Value being applied does not have a function type
```

The IDE interface includes a menu bar (File, Edit, Search, Markers, Folding, View, Utilities, Macros, Plugins, Help), a toolbar, a file browser on the right showing the project structure, and a status bar at the bottom with '5.1 (43/414)' and '(isabelle,sidekick,UTF-8-Isabelle) - - - UG 123Mb 3:25 PM'.

2012 (stable)

The screenshot shows the Isabelle/ML editor interface. The main window displays the source code for a theory named 'Seq'. The code defines a datatype for finite sequences and functions for concatenation and reversal. A yellow highlight is under the 'where' clause of the 'conc' function. A tooltip for the 'Seq' constant is visible. The right sidebar shows a tree view of the theory's structure. The bottom status bar displays the termination order: `"(\p. size (fst p)) < *mlex* {}"`.

```
header {* Finite sequences *}

theory Seq
imports Main
begin

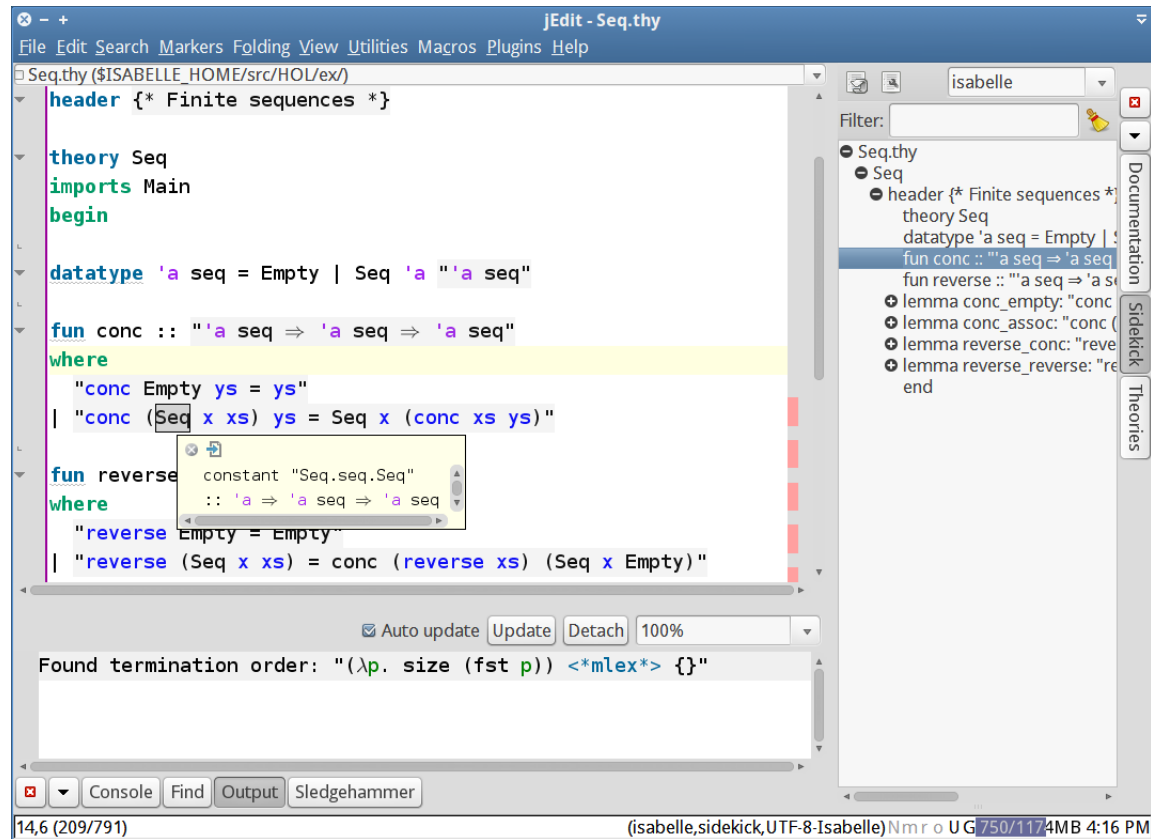
datatype 'a seq = Empty | Seq 'a "'a seq"

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

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

Found termination order: `"(\p. size (fst p)) < *mlex* {}"`

2013 (stable, rich client)



2014 (stable, filthy rich client)

```
header {* 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

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

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

Seq.thy (SISABELLE_HOME/src/HOL/ex/)

isabelle

Seq.thy

- Seq
- header {* Finite sequences *}
- theory Seq
- datatype 'a seq = Empty | Seq
- 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 = conc xs (conc ys zs)"
- lemma reverse_conc: "reverse (conc xs ys) = conc (reverse ys) (Seq x Empty)"
- lemma reverse_reverse: "reverse (reverse xs) = xs"

13,39 (203/791) (isabelle,sidekick,UTF-8-Isabelle)N m r o U C 162/853MB 11:05

jEdit 5.1

What is jEdit?

Logo: 

Website: <http://www.jedit.org>

User's guide: <http://www.jedit.org/users-guide>

Official blurb:

“jEdit is a mature programmer’s text editor with hundreds [...] of person-years of development behind it. [...] The jEdit core, together with a large collection of plugins is maintained by a world-wide developer team.”

Vox populi:

“The legendary jEdit text editor.”

Notable jEdit features

- multiple views, buffers, text areas
- dockable windows
- status bar and action bar
- advanced file management: browser, URLs, reload events
- advanced selections: range, rectangular, tall-caret, multi-selections
- copy-paste registers
- persistent markers
- advanced search-replace: hypersearch, multi-file search, search bar
- bracket matching and block selection
- folding and narrowing
- macros and BeanShell scripting
- open-ended plugin interface for JVM languages (Java, Scala)

Notable jEdit plugins

- Console: BeanShell, System shell, Scala toplevel
- Highlight: regular expression “text markers”
- SideKick: tree view
- Navigator (Isabelle2014)
- Isabelle
 - but: Isabelle/jEdit is not just another jEdit plugin
 - all-inclusive [application bundle](#)
 - different implementation language: [Scala](#) instead of Java
 - different intention: [Prover IDE](#) instead of plain text editor

**Isabelle/jEdit
in Isabelle2013-2**

Prover IDE functionality

- **File-system access:** mixed-platform notation; Isabelle settings from environment; hyperlinks for formal file references
- **Text buffers and theories:** *Theories* panel; management of editor buffers versus PIDE document-nodes; editor perspective as hint for theory processing; rendering of semantic markup
- **Prover output:** in-source messages; gutter area; text overview area; *Output* panel as imitation of old-style terminal
- **Tooltips and hyperlinks:** C-hover idiom to ask for more information; nested tooltips with same recursive structure (including hyperlinks); detachable tooltips and *Info* windows
- **Text completion:** completion tables from outer syntax and Isabelle symbols; completion popups with minimal key event handling; explicit or implicit completion

- **Isabelle symbols:** custom encoding; custom font with all required glyphs; various input methods, including *Symbols* panel; control symbols for change of font style
- **Automatically tried tools:** methods, nitpick, quickcheck, sledgehammer, solve-direct
- **Sledgehammer:** GUI control panel
- **Find theorems:** GUI control panel

Miscellaneous tools

- **SikeKick**: NEWS file; alternative parser `isabelle-markup` for PIDE document content
- **Timing**: separate panel to analyze command transactions
- **Isabelle/Scala console**: direct access to running PIDE front-end
- **Low-level output**: side-channels for system diagnostics

Isabelle/jEdit NEWS

Forthcoming Isabelle2014 (August 2014)

- syntactic and semantic **completion**, including **spell-checking**
- editor **navigation** (like web browser)
- **auxiliary files** within the document model
 - Isabelle/ML files, e.g. direct editing of Isabelle/HOL
 - **Standard ML** files: IDE for SML'97, independently of theory and proof development
- systematic nesting of sub-languages via **text cartouches**
⟨funny ⟨quotes ⟨with⟩ arbitrary ⟨nesting⟩⟩⟩
- improved platform support: **Windows, Mac OS X**
but: continued **decay of Linux** desktop environments

Syntactic and semantic completion

Completion

Goal: “Do what I mean” for the Prover IDE

Problems:

- timing of GUI events: keyboard input, mouse, popups
→ potential **loss of events**, **deadlocks**, confusing display
- need for adequate information from editor and prover
→ asynchronous document-model opens new possibilities,
but also challenges: **non-determinism**

Approach:

- interpretation of GUI events wrt. semantic **completion context**
- particular completion information from prover,
e.g. **failed name lookups**

Applications in Isabelle/jEdit

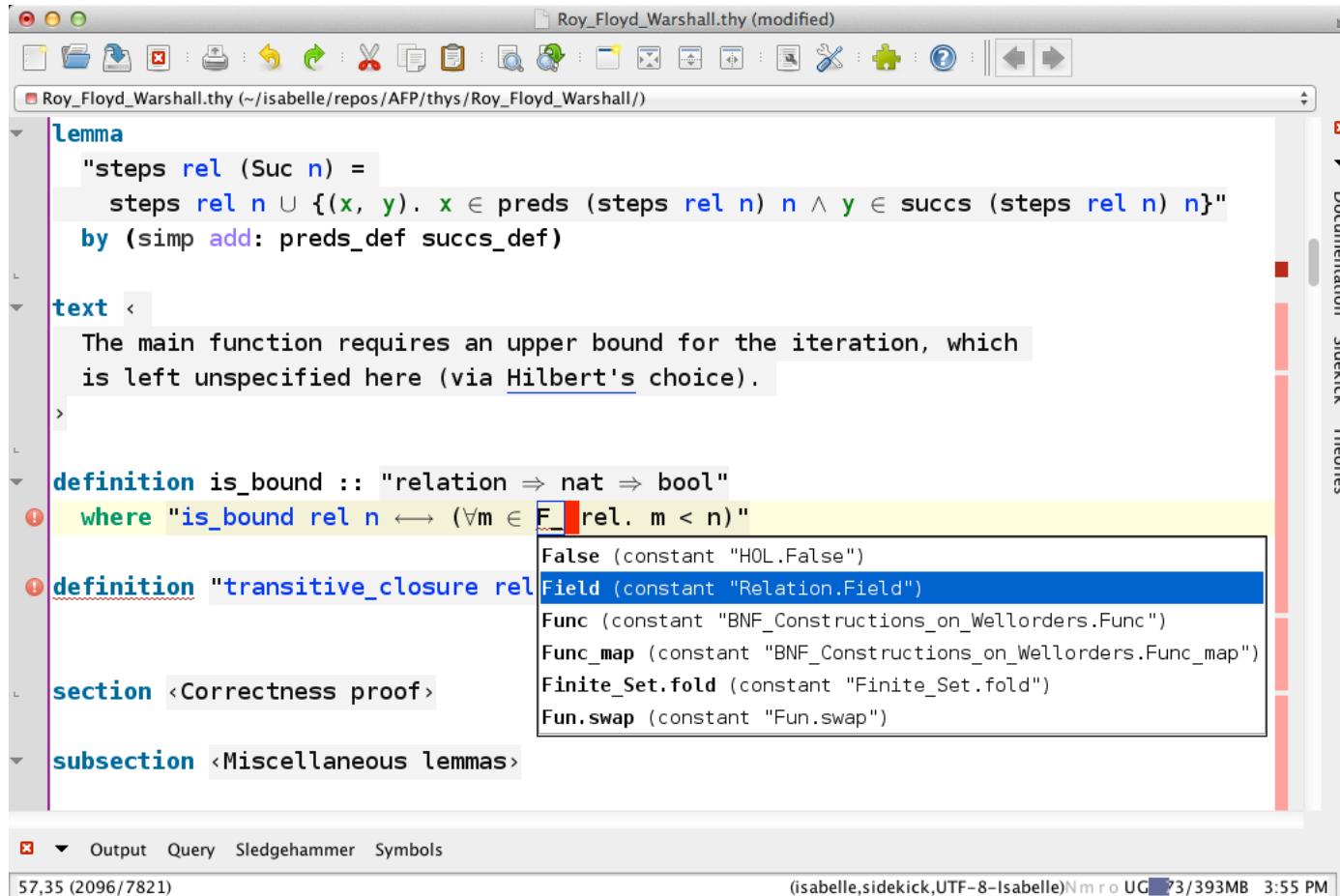
Syntactic completion:

- built-in templates (quotations, antiquotations)
- outer syntax keywords
- Isabelle symbols ($-->$ vs. \longrightarrow)

Semantic completion:

- name-space entries
- file-system paths
- spell-checking

Example: spell-checking and name completion



The screenshot shows a window titled "Roy_Floyd_Warshall.thy (modified)" with a toolbar and a sidebar. The code editor contains the following text:

```
Lemma
  "steps rel (Suc n) =
    steps rel n ∪ {(x, y). x ∈ preds (steps rel n) n ∧ y ∈ succs (steps rel n) n}"
  by (simp add: preds_def succs_def)

text <
  The main function requires an upper bound for the iteration, which
  is left unspecified here (via Hilbert's choice).
  >

definition is_bound :: "relation ⇒ nat ⇒ bool"
  where "is_bound rel n ↔ (∀m ∈ F rel. m < n)"

definition "transitive_closure rel"

section <Correctness proof>

subsection <Miscellaneous lemmas>
```

A completion popup is visible over the variable 'F' in the definition of 'is_bound'. The popup lists the following options:

- False (constant "HOL.False")
- Field (constant "Relation.Field")
- Func (constant "BNF_Constructions_on_Wellorders.Func")
- Func_map (constant "BNF_Constructions_on_Wellorders.Func_map")
- Finite_Set.fold (constant "Finite_Set.fold")
- Fun.swap (constant "Fun.swap")

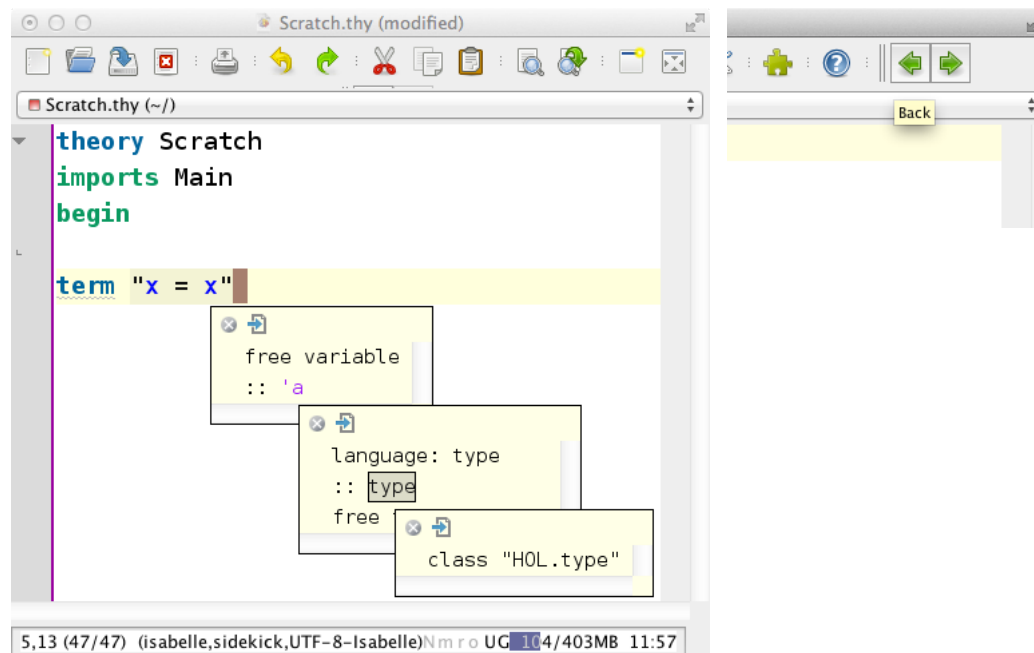
The status bar at the bottom shows "57,35 (2096/7821)" and "(isabelle,sidekick,UTF-8-Isabelle)Nm r o UG 73/393MB 3:55 PM".

Editor navigation

Editor navigation

Approach:

browsing PIDE document content via [tooltips](#) and [hyperlinks](#)



Auxiliary files within the document model

Document blobs

Principle: IDE manages collection of sources and results of processing

So far:

- acyclic graph of **document nodes** (theories)
- linear chain of **command spans** within each node

Now also:

- collection of **document blobs** (uninterpreted byte vectors)
- **load commands** refer to blobs in value-oriented manner:
 no direct file-system access

Applications in Isabelle/jEdit

Isabelle/ML:

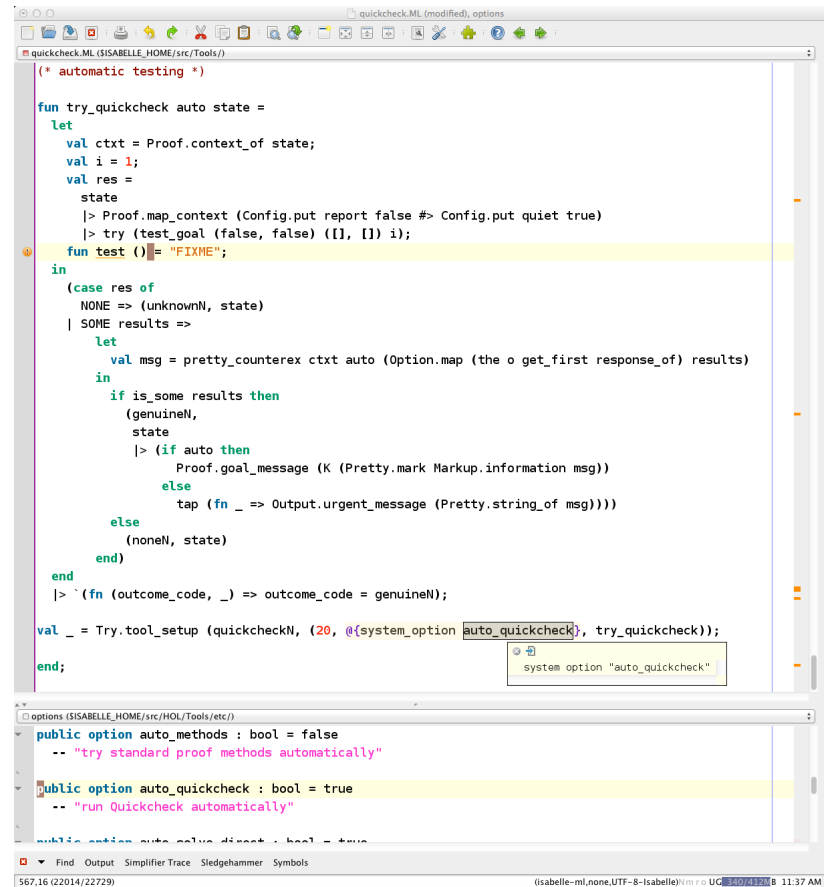
- load command **ML_file** for .ML files
- seamless editing of theories together with ML modules
- special tricks (via asynchronous print functions) to allow editing of Isabelle/HOL with only 4 GB

Official Standard ML:

- load command **SML_file** for .sml and .sig files
- commands **SML_import**, **SML_export**
for exchange of toplevel ML bindings

→ IDE for SML'97, with or without Isabelle/ML library

Example: editing Isabelle/HOL ML tools



```
(* automatic testing *)

fun try_quickcheck auto state =
  let
    val ctxt = Proof.context_of state;
    val i = 1;
    val res =
      state
      |> Proof.map_context (Config.put report false #> Config.put quiet true)
      |> try (test_goal (false, false) ([], [])) i;
  in
    fun test () = "FIXME";
  in
    (case res of
     NONE => (unknownN, state)
    | SOME results =>
      let
        val msg = pretty_counterex ctxt auto (Option.map (the o get_first response_of) results)
      in
        if is_some results then
          (genuineN,
           state
           |> (if auto then
              Proof.goal_message (K (Pretty.mark Markup.information msg))
            else
              tap (fn _ => Output.urgent_message (Pretty.string_of msg))))
        else
          (noneN, state)
        end)
      end)
    |> `(fn (outcome_code, _) => outcome_code = genuineN);
  val _ = Try.tool_setup (quickcheckN, (20, @system_option auto_quickcheck), try_quickcheck);
end;
```

```
options (SISABELLE_HOME/src/HOL/Tools/etc)
public option auto_methods : bool = false
  -- "try standard proof methods automatically"
public option auto_quickcheck : bool = true
  -- "run Quickcheck automatically"
public option auto_value_dissect : bool = true
```

system option "auto_quickcheck"

Conclusions

Conclusions

PIDE in 2014:

- Numerous increments towards full-scale Prover IDE.
- More and more **integration** of **development** in the **environment** of Isabelle: theories, documents, add-on tools, embedded languages.
- The more it advances, the higher the ambitions.

Ultimate challenge:

Introducing genuine interaction into ITP

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