

Paral-ITP Front-End Technologies

Makarius Wenzel
Univ. Paris-Sud, LRI

July 2014



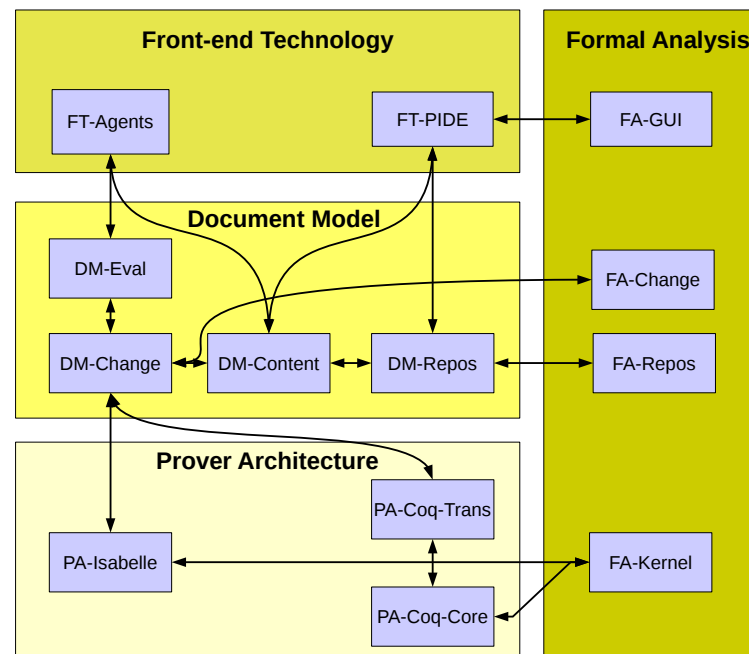
Project **Paral-ITP** meeting
ANR-11-INSE-001

Overview

Relevant work packages

FT-PIDE: Concrete **Prover IDE implementations**
(integration of prover with editor front-end)

FT-Agents: Tools for **augmented prover interaction**
(automated provers and disprovers, add-on tools and languages)



Papers and presentations (VSL 2014)

Conference papers:

- M. Wenzel: *Asynchronous User Interaction and Tool Integration in Isabelle/PIDE*. ITP 2014, Springer LNCS 8558.
- D. Matichuk, M. Wenzel, T. Murray: *An Isabelle Proof Method Language*. ITP 2014, Springer LNCS 8558.

Workshop presentations:

- M. Wenzel: *Tutorial: Isabelle/jEdit for seasoned Isabelle users*. Isabelle Workshop 2014, Vienna, July 2014.
- M. Wenzel: *Isabelle/jEdit NEWS*. Isabelle Workshop 2014, Vienna, July 2014.
- M. Wenzel: *System description: Isabelle/jEdit in 2014*. UITP 2014, Vienna, July 2014.

Releases

Isabelle2013-2 (December 2013)

- Relaunch of Isabelle2013-1 (November 2013).
- Too little serious testing by actual users!

Isabelle2014 (August 2014)

- <http://isabelle.in.tum.de/website-Isabelle2014-RC0>
- final deliverable for Paral-ITP PA/FT
- PIDE as “filthy rich client” application
- syntactic and semantic completion, including spell-checking
- editor navigation (like web browser)
- auxiliary files within the document model: Isabelle/ML, SML'97
- systematic nesting of sub-languages via text cartouches:
⟨funny ⟨quotes ⟨with⟩ arbitrary ⟨nesting⟩⟩⟩
- improved Windows support (MikTeX)

Asynchronous print functions

Asynchronous print functions

Background: READ-EVAL-PRINT within PIDE document-model

Observations:

- cumulative PRINT operations consume more time than EVAL
- PRINT depends on user perspective
- PRINT may diverge or fail
- PRINT augments results without changing proof state
- many different PRINTs may be run independently

Approach:

- manage forked PRINT tasks, depending on **document perspective**
- optional **persistence** within document-model
- optional arguments for PRINT: **document overlays**

Applications in Isabelle/jEdit

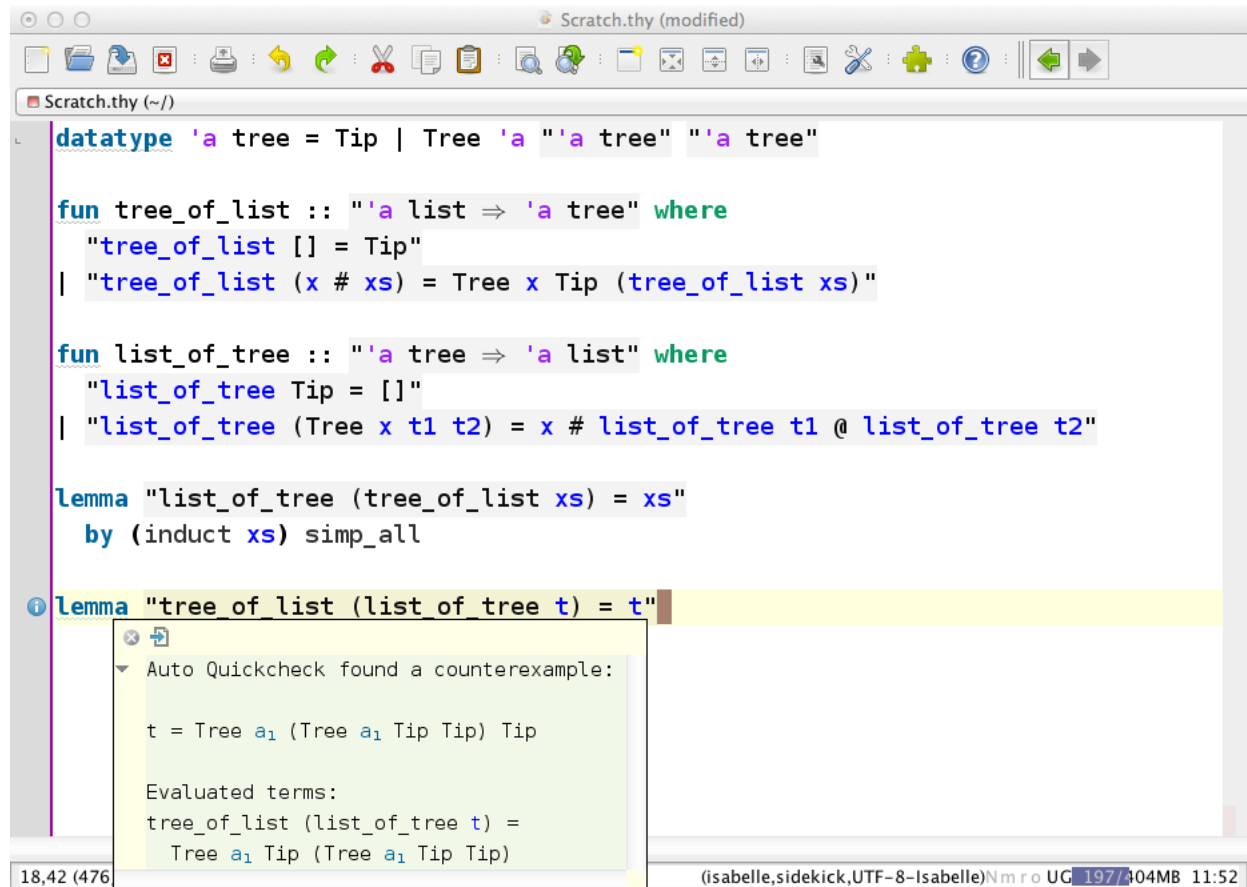
Implicit print functions:

- print proof state
(printing proof states is slower than most proof steps)
- automatically tried tools
(provers, disprovers, counter-examples)

Document overlays:

- Query panel
(find theorems, find constants, print context)
- Sledgehammer panel
(conventional GUI front-end for asynchronous tool integration)

Example: automatically tried tools



```
Scratch.thy (modified)
Scratch.thy (~/)
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

i 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)
18,42 (476) (isabelle,sidekick,UTF-8-Isabelle)N m r o UG 197/404MB 11:52
```

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 theorem prover interface with a code editor and a completion menu. 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>
```

The completion menu is open over the variable 'F' in the lambda expression. It 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")

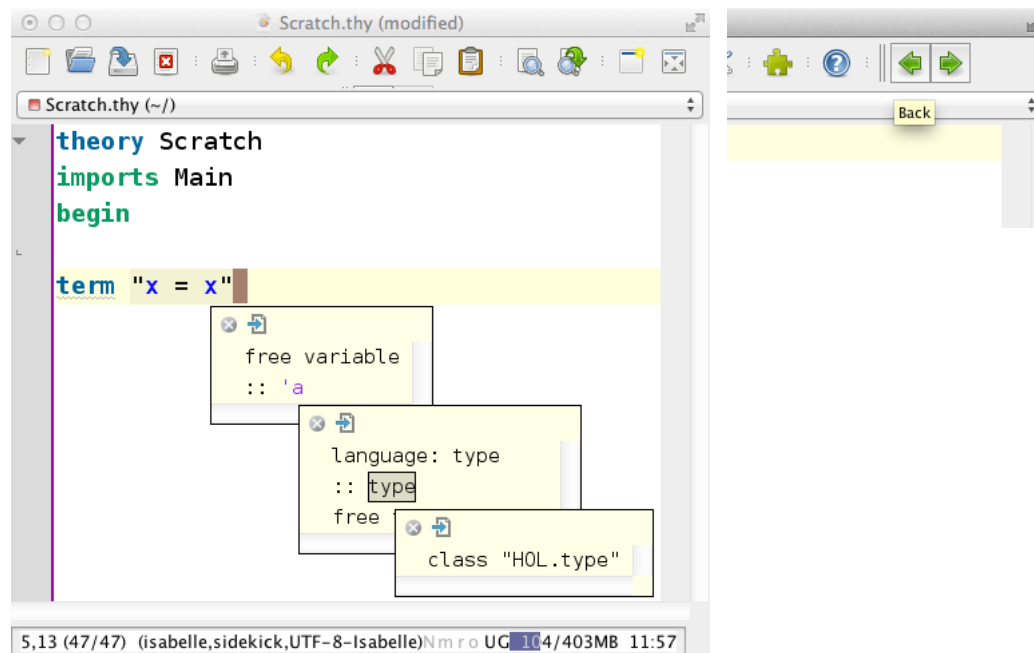
The status bar at the bottom shows "57,35 (2096/7821)" on the left and "(isabelle,sidekick,UTF-8-Isabelle)NmroUG 73/393MB 3:55 PM" on the right.

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:

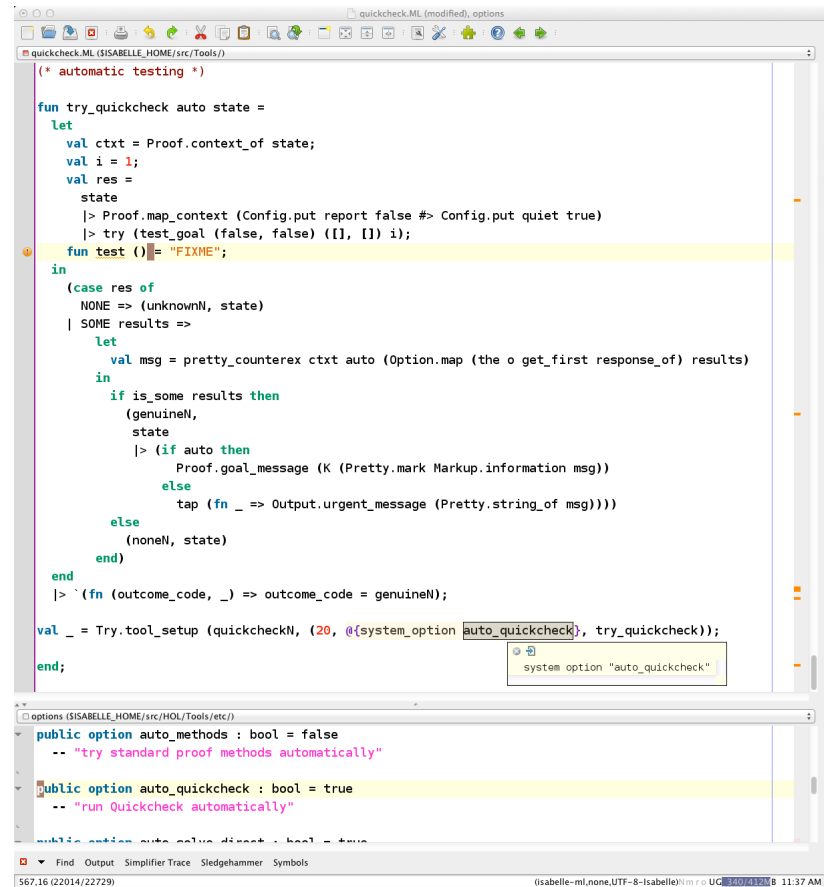
- load command **ML_file** for .ML files
- seamless editing of theories together with ML modules
- special tricks (via asynchronous print functions) to allow editing of Isabelle/HOL with only 4 GB

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;
```

system option "auto_quickcheck"

```
public option auto_methods : bool = false
  -- "try standard proof methods automatically"

public option auto_quickcheck : bool = true
  -- "run Quickcheck automatically"

public option auto_solve_dissect : bool = true
```

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