Integrating Computation and Deduction in the \texttt{ISAC}-System

Projektpraktikum: Introducing Isabelle’s Contexts

Mathias Lehnfeld

Vienna University of Technology
Institute of Computer Languages

June 1, 2011
1. Introduction: Isabelle and ISAC

2. Contributions of the project to ISAC
   Isabelle’s Contexts, advantages and use
   Redesign of ISAC using contexts
   Improvement of functional code

3. Problems encountered in the project

4. Summary
Isabelle and ISAC

The task of this “Projektpraktikum” (6 ECTS) was to

- study the concept of “context” in the theorem prover Isabelle from TU Munich
- study basic concepts of the math assistant ISAC from TU Graz
- redesign ISAC with respect to contexts
  - use contexts for type inference of user input
  - handle preconditions of specifications
  - clarify the transfer of context data from sub-programs to the calling program
- introduce contexts to ISAC according to the new design
- use the coding standards of Isabelle2011 for new code.
Isabelle and ISAC

The task of this “Projektpraktikum” (6 ECTS) was to

• study the concept of “context” in the theorem prover Isabelle from TU Munich
• study basic concepts of the math assistant ISAC from TU Graz
• redesign ISAC with respect to contexts
  • use contexts for type inference of user input
  • handle preconditions of specifications
  • clarify the transfer of context data from sub-programs to the calling program
• introduce contexts to ISAC according to the new design
• use the coding standards of Isabelle2011 for new code.
Isabelle and ISAC

The task of this “Projektpraktikum” (6 ECTS) was to

- study the concept of “context” in the theorem prover \textbf{Isabelle} from TU Munich
- study basic concepts of the math assistant \textbf{ISAC} from TU Graz
- redesign \textbf{ISAC} with respect to contexts
  - use contexts for type inference of user input
  - handle preconditions of specifications
  - clarify the transfer of context data from sub-programs to the calling program
- introduce contexts to \textbf{ISAC} according to the new design
- use the coding standards of Isabelle2011 for new code.
Computation and Deduction in a Lucas-Interpreter

Isac
specify
calculate
calctree
rewriter
context
context
context
Frontend

Lucas-interpreter

Isabelle
context
library
combinators
match
parse
pretty_print
theory management

specification
program
theories
Outline

1 Introduction: Isabelle and ISAC

2 Contributions of the project to ISAC
   Isabelle’s Contexts, advantages and use
   Redesign of ISAC using contexts
   Improvement of functional code

3 Problems encountered in the project

4 Summary
Advantages of Isabelle’s Contexts

Isabelle’s context replaced theories because . . . :

• theories are static containers of all logical data
• contexts are *dynamic* containers of logical data:
  • functions for storing and retrieving various logical data
  • functions for type inference
  • provide data for Isabelle’s automated provers
• allow to conform with scopes for subprograms.
Isabelle’s context mechanism

structure ContextData = Proof_Data
  (type T = term list
   fun init _ = []);

fun insert_assumptions asms =
  ContextData.map (fn xs => distinct (asms @ xs));

fun get_assumptions ctxt = ContextData.get ctxt;

val declare_constraints :
  term -> Proof.context -> Proof.context
Usage of Contexts

fun transfer_asms_from_to from_ctxt to_ctxt =
    let
        val to_vars = get_assumptions to_ctxt |> map vars |> flat
        fun transfer [] to_ctxt = to_ctxt
            | transfer (from_asm::fas) to_ctxt =
                if inter op = (vars from_asm) to_vars = []
                    then transfer fas to_ctxt
                    else transfer fas (insert_assumptions [from_asm] to_ctxt)
    in transfer (get_assumptions from_ctxt) to_ctxt end

fun parse thy str =
    (let val t = (typ_a2real o numbers_to_string)
        (Syntax.read_term_global thy str)
        in SOME (cterm_of thy t) end)
    handle _ => NONE;

fun parseNEW ctxt str =
    SOME (Syntax.read_term ctxt str |> numbers_to_string)
    handle _ => NONE;
ISAC: Computation & Deduction
Lehnfeld

Outline

1. Introduction: Isabelle and ISAC
2. Contributions of the project to ISAC
   - Isabelle’s Contexts, advantages and use
   - Redesign of ISAC using contexts
   - Improvement of functional code
3. Problems encountered in the project
4. Summary
Redesign of \textit{ISAC} using contexts
Deduction simplifies computation

(some) assumptions

\[
\text{solve } \left( \frac{x}{x^2 - 6x + 9} - \frac{1}{x^2 - 3x} = \frac{1}{x}, x \right)
\]

\[
\frac{3 + -1* x + x^2}{9*x + -6*x^2 + x^3} = \frac{1}{x}
\]

\[
x \neq 3 \land x \neq 0
\]

\[
(3 + -1* x + x^2) * x = 1 * (9 * x + -6 * x^2 + x^3)
\]

\[
\text{solve } ((3 + -1* x + x^2) * x = 1 * (9 * x + -6 * x^2 + x^3), x)
\]

\[
(3 + -1* x + x^2) * x - (9 * x + -6 * x^2 + x^3) = 0
\]

\[
-6 * x + 5 * x^2 = 0
\]

\[
\text{solve } (-6 * x + 5 * x^2 = 0, x)
\]

\[
[x = 0, x = \frac{6}{5}]
\]

\[
x = 0 \land x = \frac{6}{5}
\]

Check Elementwise Assumptions : \(x \neq 0 \land x = 0\)

\[
[x = 0, x = \frac{6}{5}]
\]

\[
[x = \frac{6}{5}]
\]

\[
[x = \frac{6}{5}]
\]
More “deduction”, less “computation”

Script Solve_root_equation (e_e::bool) (v_v::real) =
  (let e_e = ((Try (Rewrite_Set norm_equation False))
              (Try (Rewrite_Set Test_simplify False))) e_e;
   (L_L::bool list) =
     (SubProblem (Test’,
                 [linear,univariate,equation,test]
                 [Test,solve_linear])
      [BOOL e_e, REAL v_v])
  in Check_elementwise L_L {(v_v::real). Assumptions})

“Deductive” part of Lucas-Interpretation relives the
“computational” part: one statement becomes obsolete!
Advantages of the redesign:

- **type inference by *local* contexts**
  
  now user-input without type constraints!

- consistent handling of logical data
  
  - preconditions and partiality conditions in contexts
  
  - transfer of context data into subprograms clarified
  
  - transfer of context data from subprograms clarified

  now some statements become obsolete.

Now Lucas-interpretation shifts efforts from “computation” further to “deduction”.
Redesign of ISAC using contexts

Advantages of the redesign:

- type inference by *local* contexts
  now user-input without type constraints!
- consistent handling of logical data
  - preconditions and partiality conditions in contexts
  - transfer of context data into subprograms clarified
  - transfer of context data from subprograms clarified
  now some statements become obsolete.

Now Lucas-interpretation shifts efforts from “computation” further to “deduction”.
Redesign of \textit{ISAC} using contexts

Advantages of the redesign:

- type inference by local contexts
  now user-input without type constraints!
- consistent handling of logical data
  - preconditions and partiality conditions in contexts
  - transfer of context data into subprograms clarified
  - transfer of context data from subprograms clarified

now some statements become obsolete.

Now Lucas-interpretation shifts efforts from “computation” further to “deduction”.
Redesign of ISAC using contexts

Advantages of the redesign:

- type inference by *local* contexts
  now user-input without type constraints!
- consistent handling of logical data
  - preconditions and partiality conditions in contexts
  - transfer of context data into subprograms clarified
  - transfer of context data from subprograms clarified

now some statements become obsolete.

Now Lucas-interpretation shifts efforts from “computation” further to “deduction”.
Redesign of \textit{ISAC} using contexts

Advantages of the redesign:

- type inference by \textit{local} contexts
  now user-input without type constraints!
- consistent handling of logical data
  - preconditions and partiality conditions in contexts
  - transfer of context data into subprograms clarified
  - transfer of context data from subprograms clarified
  now some statements become obsolete.

Now Lucas-interpretation shifts efforts from “computation” further to “deduction”.
1 Introduction: Isabelle and ISAC

2 Contributions of the project to ISAC
   Isabelle’s Contexts, advantages and use
   Redesign of ISAC using contexts
   Improvement of functional code

3 Problems encountered in the project

4 Summary
Improvement of functional code

- **code conventions**: Isabelle2011 published coding standards first time

- **combinators**: Isabelle2011 introduced a library containing the following combinators:

```plaintext
val |> : 'a * ('a -> 'b) -> 'b
val |-> : ('c * 'a) * ('c -> 'a -> 'b) -> 'b
val |>> : ('a * 'c) * ('a -> 'b) -> 'b * 'c
val ||> : ('c * 'a) * ('a -> 'b) -> 'c * 'b
val ||>> : ('c * 'a) * ('a -> 'd * 'b) -> ('c * 'd) * 'b
val #> : ('a -> 'b) * ('b -> 'c) -> 'a -> 'c
val #-> : ('a -> 'c * 'b) * ('c -> 'b -> 'd) -> 'a -> 'd
val #>> : ('a -> 'c * 'b) * ('c -> 'd) -> 'a -> 'd * 'b
val ##> : ('a -> 'c * 'b) * ('b -> 'd) -> 'a -> 'c * 'd
val ##> : ('a -> 'c * 'b) * ('b -> 'e * 'd) -> 'a -> ('c * 'e) * 'd
```
Example with combinators

```haskell
fun prep_ori [] _ _ = ([], e_ctxt)
  | prep_ori fmz thy pbt = 
    let
      val ctxt = ProofContext.init_global thy
      |> fold declare_constraints fmz
      val ori =
        map (add_field thy pbt o split_dts o the o parseNEW ctxt) fmz
        |> add_variants
      val maxv = map fst ori |> max
      val maxv = if maxv = 0 then 1 else maxv
      val oris = coll_variants ori
      |> map (replace_0 maxv |> apfst)
      |> add_id
      |> map flattup
    in (oris, ctxt) end;

...which probably can be further polished.
Problems encountered in the project

- new Isabelle release in February 2011: update ISAC first
- lines of code (LOC) in Isabelle and ISAC

\[
\begin{align*}
\text{src/} & : \quad 1700 \text{ k LOC} \\
\text{src/Pure/} & : \quad 70 \text{ k LOC} \\
\text{src/Provers/} & : \quad 8 \text{ k LOC} \\
\text{src/Tools/} & : \quad 800 \text{ k LOC} \\
\text{src/Tools/isac/} & : \quad 37 \text{ k LOC} \\
\text{src/Tools/isac/Knowledge} & : \quad 16 \text{ k LOC} \\
\end{align*}
\]
- changes scattered throughout the code (→ grep)
- documentation of Isabelle very “technical” (no API)
- documentation of ISAC not up to date
Problems encountered in the project

- new Isabelle release in February 2011: update ISAC first
- lines of code (LOC) in Isabelle and ISAC

```
src/               1700  k LOC
src/Pure/          70   k LOC
src/Provers/       8    k LOC
src/Tools/         800  k LOC
src/Tools/isac/    37   k LOC
src/Tools/isac/Knowledge  16  k LOC
```
- changes scattered throughout the code (→ grep)
- documentation of Isabelle very “technical” (no API)
- documentation of ISAC not up to date
Problems encountered in the project

- new Isabelle release in February 2011: update ISAC first
- lines of code (LOC) in Isabelle and ISAC

```plaintext
<table>
<thead>
<tr>
<th>Directory</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>src/</td>
<td>1700k</td>
</tr>
<tr>
<td>src/Pure/</td>
<td>70k</td>
</tr>
<tr>
<td>src/Provers/</td>
<td>8k</td>
</tr>
<tr>
<td>src/Tools/</td>
<td>800k</td>
</tr>
<tr>
<td>src/Tools/isac/</td>
<td>37k</td>
</tr>
<tr>
<td>src/Tools/isac/Knowledge</td>
<td>16k</td>
</tr>
</tbody>
</table>
```

- changes scattered throughout the code (→ grep)
- documentation of Isabelle very “technical” (no API)
- documentation of ISAC not up to date
Problems encountered in the project

- new Isabelle release in February 2011: update \textit{ISAC} first
- lines of code (LOC) in Isabelle and \textit{ISAC}

\begin{verbatim}
src/ 1700 k LOC
src/Pure/ 70 k LOC
src/Provers/ 8 k LOC
src/Tools/ 800 k LOC
src/Tools/isac/ 37 k LOC
src/Tools/isac/Knowledge 16 k LOC
\end{verbatim}

- changes scattered throughout the code (→ grep)
- documentation of Isabelle very “technical” (no API)
- documentation of \textit{ISAC} not up to date
Problems encountered in the project

- new Isabelle release in February 2011: update ISAC first
- lines of code (LOC) in Isabelle and ISAC
  - src/ 1700 k LOC
  - src/Pure/ 70 k LOC
  - src/Provers/ 8 k LOC
  - src/Tools/ 800 k LOC
  - src/Tools/isac/ 37 k LOC
  - src/Tools/isac/Knowledge 16 k LOC
- changes scattered throughout the code (→ grep)
- documentation of Isabelle very “technical” (no API)
- documentation of ISAC not up to date
The project succeeded in all goals:
- implemented Isabelle’s contexts in \textit{TSAC} such that
- user input requires no type constraints anymore
- consistent logical data is prepared for Isabelle’s provers

The course of the project was close to the plan:
- faster in writing new code
- slower in integrating the code into \textit{TSAC}

The project provided essential prerequisites for further development of the Lucas-interpreter.
The project succeeded in all goals:

- implemented Isabelle’s contexts in ISAC such that
- user input requires no type constraints anymore
- consistent logical data is prepared for Isabelle’s provers

The course of the project was close to the plan:

- faster in writing new code
- slower in integrating the code into ISAC

The project provided essential prerequisites for further development of the Lucas-interpreter.
The project succeeded in all goals:

- implemented Isabelle’s contexts in ISAC such that
- user input requires no type constraints anymore
- consistent logical data is prepared for Isabelle’s provers

The course of the project was close to the plan:

- faster in writing new code
- slower in integrating the code into ISAC

The project provided essential prerequisites for further development of the Lucas-interpreter.