Lucas Interpretation from Programmers’ Perspective. (Work in Progress Report)

Walther Neuper

University of Technology Graz, Austria

ThEdu’18 at FLoC, Oxford
July 18, 2018
Outline

1. Towards Isabelle/ISAC

2. Adaptation of Isabelle’s function package
   - Introduce specific tactics
   - Interpret tactics as break-points
   - New appearance in Isabelle
   - Note: specifications are explicit

3. Use for verified applied mathematics?
Towards Isabelle/\texttt{ISAC}

- Shift \texttt{ISAC}'s programming language to Isabelle's function package
  - Shift \texttt{ISAC}'s Worksheet to a window in Isabelle/jEdit:
    - adapt Isabelle's parsers for outer syntax to \texttt{ISAC}'s calculation (i.e. \texttt{ISAC}'s forward proof)
    - Provide \texttt{ISAC}'s calculation as a pattern
    - Let \texttt{ISAC}'s Lucas-Interpreter work in Isabelle's proof state . . .
  - Introduce multi-user session management:
    - Re-use code of the new Isabelle server
    - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .
  - Develop a graphical formula editor for Isabelle/jEdit
  - Provide authoring tools for specifications, . . .
- . . .
Towards Isabelle/ISAC

- Shift ISAC’s programming language to Isabelle’s function package
- Shift ISAC’s Worksheet to a window in Isabelle/jEdit:
  - adapt Isabelle’s parsers for outer syntax to ISAC’s calculation (i.e. ISAC’s forward proof)
  - Provide ISAC’s calculation as a pattern
  - Let ISAC’s Lucas-Interpreter work in Isabelle’s proof state . . .
- Introduce multi-user session management:
  - Re-use code of the new Isabelle server
  - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .
- Develop a graphical formula editor for Isabelle/jEdit
- Provide authoring tools for specifications, . . .
- . . .
Towards Isabelle/\textsc{Isac}

- Shift \textsc{Isac}'s programming language to Isabelle's function package
- Shift \textsc{Isac}'s Worksheet to a window in Isabelle/jEdit:
  - adapt Isabelle's parsers for outer syntax to \textsc{Isac}'s calculation (i.e. \textsc{Isac}'s forward proof)
  - Provide \textsc{Isac}'s calculation as a pattern
  - Let \textsc{Isac}'s Lucas-Interpreter work in Isabelle's proof state . . .
- Introduce multi-user session management:
  - Re-use code of the new Isabelle server
  - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .
- Develop a graphical formula editor for Isabelle/jEdit
- Provide authoring tools for specifications, . . .
- . . .
Towards Isabelle/\textit{ISAC}

- Shift \textit{ISAC}'s programming language to Isabelle's function package

- Shift \textit{ISAC}'s Worksheet to a window in Isabelle/jEdit:
  - adapt Isabelle’s parsers for outer syntax to \textit{ISAC}'s calculation (i.e. \textit{ISAC}'s forward proof)
  - Provide \textit{ISAC}'s calculation as a pattern
  - Let \textit{ISAC}'s Lucas-Interpreter work in Isabelle’s proof state . . .

- Introduce multi-user session management:
  - Re-use code of the new Isabelle server
  - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .

- Develop a graphical formula editor for Isabelle/jEdit
  - Provide authoring tools for specifications, . . .
  - . . .
Towards Isabelle/\textit{ISAC}

- Shift \textit{ISAC}'s programming language to Isabelle's function package

- Shift \textit{ISAC}'s Worksheet to a window in Isabelle/jEdit:
  - adapt Isabelle's parsers for outer syntax to \textit{ISAC}'s calculation (i.e. \textit{ISAC}'s forward proof)
  - Provide \textit{ISAC}'s calculation as a pattern
  - Let \textit{ISAC}'s Lucas-Interpreter work in Isabelle's proof state . . .

- Introduce multi-user session management:
  - Re-use code of the new Isabelle server
  - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .

- Develop a graphical formula editor for Isabelle/jEdit
- Provide authoring tools for specifications, . . .
- . . .
Towards Isabelle/\textit{ISAC}

- Shift \textit{ISAC}'s programming language to Isabelle's function package
- Shift \textit{ISAC}'s Worksheet to a window in Isabelle/jEdit:
  - adapt Isabelle's parsers for outer syntax to \textit{ISAC}'s calculation (i.e. \textit{ISAC}'s forward proof)
  - Provide \textit{ISAC}'s calculation as a pattern
  - Let \textit{ISAC}'s Lucas-Interpreter work in Isabelle's proof state . . .
- Introduce multi-user session management:
  - Re-use code of the new Isabelle server
  - Insert session management close to Isabelle/jEdit (parallel to the Isabelle plugin) . . .
- Develop a graphical formula editor for Isabelle/jEdit
- Provide authoring tools for specifications, . . .
- . . .
Outline

1. Towards Isabelle/\textit{ISAC}

2. Adaptation of Isabelle’s function package
   - Introduce specific tactics
   - Interpret tactics as break-points
   - New appearance in Isabelle
   - Note: specifications are explicit

3. Use for verified applied mathematics?
Introduce specific tactics

Syntax of Isabelle’s function package:

01 definition ::= partial_function (tailrec) fun-id :: signature where program

02 program ::= ” fun-id arg = ( body ) ”

03 fun-id ::= identifier

04 arg ::= identifier

05 body ::= if bool-expr then expr else expr

06 | let assigns in expr

07 | expr

08 assigns ::= (assign ;)* assign

09 assign ::= identifier = body

10 expr ::= ( tac-expr | no-tac-expr )

The “program” is a lambda term.
Introduce specific tactics

We separate tactics from sub-terms without tactics . . .

21  tac-expr ::= tacs no-tac-expr
22  | SubProblem ( identifier, key, key ) probl-args
23  tacs ::= tactical-1 tacs
24  | tactical-2 tacs tacs
25  | tactic
26  key ::= [ ( ID , )∗ ID ]
27  ID ::= identifier (* declared as constant *)
28  probl-args ::= [ ( probl-arg , )∗ probl-arg ]
29  probl-arg ::= type-con no-tac-expr
30  type-con ::= REAL | REAL_LIST | REAL_SET
             | BOOL | BOOL_LIST

. . . and add subproblems, which allow for interactive specification.
Introduce specific tactics

And these are the tactics . . .

tactic ::= Take
   | Rewrite ID
   | Rewrite.Inst subst ID
   | Calculate op
   | Rewrite.Set ID
   | Rewrite.Inst subst ID
   | Substitute

subst ::= ( (identifier, no-tac-expr),)* (identifier, no-tac-expr)

op ::= PLUS | MINUS | TIMES | DIVIDE | POWER

. . . and tacticals (combining tactics):

tactical-1 ::= Try
   | Repeat
   | While bool-expr

tactical-2 ::= Or
   | @@
1. Towards Isabelle/ISAC

2. Adaptation of Isabelle’s function package
   - Introduce specific tactics
   - Interpret tactics as break-points
   - New appearance in Isabelle
   - Note: specifications are explicit

3. Use for verified applied mathematics?
Interpret tactics as break-points

...by “Lucas-Interpretation”:
Interpret tactics as break-points

...by “Lucas-Interpretation”:
Interpret tactics as break-points

...by “Lucas-Interpretation”:
Interpret tactics as break-points

...by “Lucas-Interpretation”:

\[ LI : \]

- program
  * location
  * environment

  \[ \text{interpret} \]

  - location
    * environment
    * calculation

  * context

  \[ \text{formula or tactic} \]
Interpret tactics as break-points

...by “Lucas-Interpretation”:
Interpret tactics as break-points

...by “Lucas-Interpretation”:

\[
\begin{align*}
&\text{computation} \\
&\text{LI}: \\
&\text{deduction}
\end{align*}
\]
Outline

1. Towards Isabelle/ISAC

2. Adaptation of Isabelle’s function package
   - Introduce specific tactics
   - Interpret tactics as break-points
   - New appearance in Isabelle
   - Note: specifications are explicit

3. Use for verified applied mathematics?
New appearance in Isabelle

```isabelle
partial_function (tailrec) biegelinie ::
  "real ⇒ real ⇒ real ⇒ (real ⇒ real) ⇒ bool list ⇒ bool"
where
  "biegelinie l q v b s =
  (let
    funs = (SubProblem (Biegelinie',
      [vonBelastungZu, Biegelinien], [Biegelinien, ausBelastung])
      [REAL q, REAL v]);
    equs = (SubProblem (Biegelinie',
      [setzeRandbedingungen, Biegelinien], [Biegelinien, setzeRandbedingungenEin])
      [BOOL_LIST funs, BOOL_LIST s]);
    cons = (SubProblem (Biegelinie', [LINEAR, system], [no_met])
      [BOOL_LIST equs, REAL_LIST [c, c_2, c_3, c_4]]);
    B = Take (lastI funs);
  B = ((Substitute cons) @@
      [Rewrite_Set_Inst [(bdv, v)] make_ratpoly_in False]) B
in B)
```

**Figure:** Compare old appearance at
Outline

1. Towards Isabelle/\texttt{ISAC}

2. Adaptation of Isabelle’s function package
   - Introduce specific tactics
   - Interpret tactics as break-points
   - New appearance in Isabelle
   - Note: specifications are explicit

3. Use for verified applied mathematics?
Specifications are explicit . . .

. . . during interactive calculation (on the old front-end):

Figure: ISAC’s front-end shows the current specification in the Problem browser at the bottom right.
Lucas Interpretation from Programmers’ Perspective. (Work in Progress Report)

Walther Neuper

Isabelle/TSAC Function package
Specific tactics
Lucas-Interpretation
New appearance
Explicit specifications

Verified applied math
Use for verified applied mathematics?

- **AFP, Isabelle’s Archive of Formal Proofs** — a growing collection *also of algorithms*!

- Upgrade AFP’s executable functions to Isabelle/Isabelle:
  - make specifications of functions explicit (pre-conditions and post-condition)
  - use Isabelle/Isabelle’s tactics
  - i.e. achieve modularity & interactivity

- Isabelle/Isabelle — a training tool in Formal Methods?

---

Use for verified applied mathematics?

- AFP, Isabelle’s Archive of Formal Proofs \(^1\) —
  — a growing collection *also of algorithms*!

- Upgrade AFP’s executable functions to Isabelle/*ISAC*:
  - make specifications of functions explicit (pre-conditions and post-condition)
  - use Isabelle/*ISAC*’s tactics
  - i.e. achieve modularity & interactivity

- Isabelle/*ISAC* — a training tool in Formal Methods?

\(^1\)https://www.isa-afp.org/
Use for verified applied mathematics?

- AFP, Isabelle’s Archive of Formal Proofs \(^1\) — a growing collection *also of algorithms*!

- Upgrade AFP’s executable functions to *Isabelle/ISAC*:
  - make specifications of functions explicit (pre-conditions and post-condition)
  - use *Isabelle/ISAC*’s tactics
  - i.e. achieve modularity & interactivity

- *Isabelle/ISAC* — a training tool in Formal Methods?

\(^1\)https://www.isa-afp.org/
Thank you for Attention!