

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/ISA C
- 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/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/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/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/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/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/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, ...
- ...

Outline

1 Towards Isabelle/ISA C

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 ...

```
41 tactic ::= Take
42         | Rewrite ID
43         | Rewrite_Inst subst ID
44         | Calculate op
45         | Rewrite_Set ID
46         | Rewrite_Inst subst ID
47         | Substitute
48 subst  ::= [ ( (identifier, no-tac-expr) ,)* (identifier, no-tac-expr) ]
49 op      ::= PLUS | MINUS | TIMES | DIVIDE | POWER
```

... and tacticals (combining tactics):

```
61 tactical-1 ::= Try
62             | Repeat
63             | While bool-expr
64 tactical-2 ::= Or (* infix *)
65             | @@ (* infix *)
```

Outline

- 1 Towards Isabelle/ISA C
- 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":



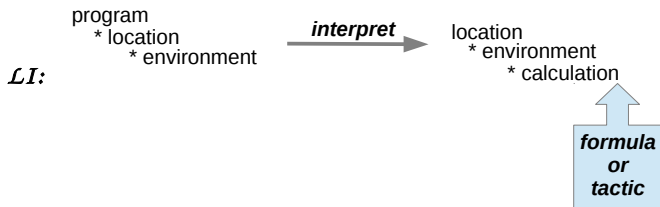
Interpret tactics as break-points

... by "Lucas-Interpretation":



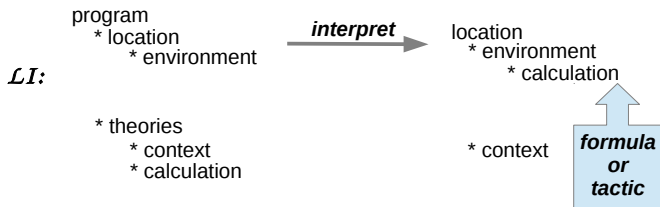
Interpret tactics as break-points

... by "Lucas-Interpretation":



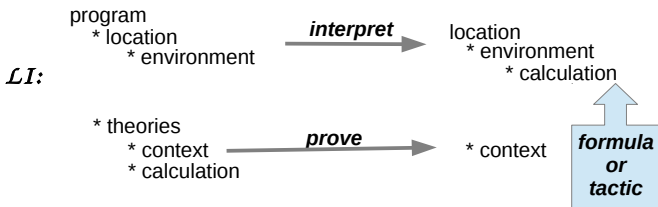
Interpret tactics as break-points

... by "Lucas-Interpretation":



Interpret tactics as break-points

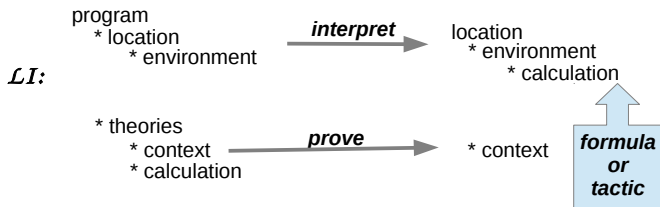
... by "Lucas-Interpretation":



Interpret tactics as break-points

... by "Lucas-Interpretation":

computation



deduction

Outline

1 Towards Isabelle/ISA C

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

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

Walther
Neuper

Isabelle/ISAC

Function
package

Specific tactics

Lucas-Interpretation

New appearance

Explicit specifications

Verified
applied math

```
partial_function (tailrec) biegeleinie ::  
  "real  $\Rightarrow$  real  $\Rightarrow$  real  $\Rightarrow$  (real  $\Rightarrow$  real)  $\Rightarrow$  bool list  $\Rightarrow$  bool"  
where  
  "biegeleinie l q v b s =  
    (let  
      funks = (SubProblem (Biegeleinie',  
        [vonBelastungZu, Biegeleinien], [Biegeleinien, ausBelastung])  
        [REAL q, REAL v]);  
      equs = (SubProblem (Biegeleinie',  
        [setzeRandbedingungen, Biegeleinien], [Biegeleinien, setzeRandbedingungenEin])  
        [BOOL_LIST funks, BOOL_LIST s]);  
      cons = (SubProblem (Biegeleinie', [LINEAR, system], [no_met])  
        [BOOL_LIST equs, REAL_LIST [c, c_2, c_3, c_4]]);  
      B = Take (lastI funks);  
      B = ((Substitute cons) @@  
        [Rewrite_Set_Inst [(bdv, v)] make_ratpoly_in False]) B  
    in B)
```

Figure: Compare old appearance at

<https://intra.ist.tugraz.at/hg/isa/file/ba408e905cce/src/Tools/isac/Knowledge/Biegeleinie.thy#l283>.

Outline

1 Towards Isabelle/ISAAC

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):

The screenshot displays the ISAC software interface with several windows open:

- Example browser:** Shows a diagram of a beam of length L with a constant distributed load q_0 . The diagram is labeled "Abb.7.50". Below it, the text reads: "Bestimme die Biegelinie für einen einseitig eingespannten Träger (Abb. 7.50) der Länge L mit konstanter Streckenlast q_0 unter Verwendung der Randbedingungen $Q(0) = q_0 \cdot L$, $M_b(L = 0, y(0) = 0, y'(0) = 0$."
- Worksheet:** Shows the mathematical steps for solving the problem:
 - $q(x) = q_0$
 - $-q(x) = -q_0$ (with a note: Rewrite("Belastung_Querkraft",""))
 - $Q'(x) = -q_0$
 - $\text{integrate}(-q_0, x)$
 - $Q(x) = \text{integral}(-q_0, x)$
 - $Q(x) = \text{integral}(-1 * q_0, x)$
 - $Q(x) = c + -1 * q_0 * x$
- Method browser:** Shows "Context On->Off".
- Theory browser:** Shows "Context On->Off" and "To Worksheet".
- Problem browser:** Shows "Context On->Off" and "Try Refine". It displays the current problem: "Problem (Biegelinie, [vonBelastungZu, Biegelinien])". Below it, a table shows the model variables:

Modell	
Given:	Streckenlast q_0 FunktionsVariable x
Where:	
Find:	Funktionen $funs''$
Relate:	

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/*ISAC*

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 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/>

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/>

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

Walther
Neuper

Isabelle/ISAAC

Function
package

Specific tactics

Lucas-Interpretation

New appearance

Explicit specifications

Verified
applied math

Thank you for Attention!