Lucas-Interpretation from Users’ Perspective

Walther Neuper

IICM, Institute for Computer Media,
University of Technology.
Graz, Austria

ThEdu: Theorem Proving Components for Educational Software
at CICM, Bialystok, Poland
July 25, 2016
1. User’s Views
   - Demo: Programmers’ View
   - Demo: Students’ View

2. Lucas-Interpretation
   - The Language
   - The Interpreter
   - Where is Interaction from?
   - Summary

3. Conclusions for Users
   - Usability for Programmers
   - Self-explaining System for Students
1 User’s Views

**Demo**: Programmers’ View
**Demo**: Students’ View

2 Lucas-Interpretation

The Language
The Interpreter
Where is Interaction from?
Summary

3 Conclusions for Users

Usability for Programmers
Self-explaining System for Students
Programmers’ View

Demonstration

Summary:

- Programming is painful presently …
  - program syntax checked as Isabelle term
  - rewrite-sets for execution compiled by hand
- … thus migration to Isabelle’s function package

- functional programs without input / output — where comes user-interaction from ???
Demonstration

Summary:

- Programming is painful presently ...
  - program syntax checked as Isabelle term
  - rewrite-sets for execution compiled by hand
- ... thus migration to Isabelle’s function package
- functional programs without input / output — where comes user-interaction from ???
Programmers’ View

Demonstration

Summary:

- Programming is painful presently . . .
  - program syntax checked as Isabelle term
  - rewrite-sets for execution compiled by hand
- . . . thus migration to Isabelle’s function package

- functional programs without input / output — where comes user-interaction from ???
Programmers’ View

Demonstration

Summary:

- Programming is painful presently . . .
  - program syntax checked as Isabelle term
  - rewrite-sets for execution compiled by hand
- . . . thus migration to Isabelle’s function package
- functional programs without input / output — where comes user-interaction from ???
1 User’s Views
   Demo: Programmers’ View
   Demo: Students’ View

2 Lucas-Interpretation
   The Language
   The Interpreter
   Where is Interaction from?
   Summary

3 Conclusions for Users
   Usability for Programmers
   Self-explaining System for Students
Demonstration

Summary: Students require these services for learning . . .

1. **check user input** automatically, **flexibly** and reliably:
   Input establishes a *proof situation* (for *automated* proving) with respect to the logical context.

2. **give explanations** on request by learners:
   All underlying mathematics knowledge is **transparent** due to the “LCF-paradigm” in Isabelle.

3. **propose a next step** if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Students’ View

Demonstration

Summary: Students require these services for learning . . .

1. check user input automatically, flexibly and reliably:
   Input establishes a proof situation (for automated proving) with respect to the logical context

2. give explanations on request by learners:
   All underlying mathematics knowledge is transparent due to the “LCF-paradigm” in Isabelle

3. propose a next step if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Students’ View

Demonstration

Summary: Students require these services for learning . . .

1. check user input automatically, flexibly and reliably:
   Input establishes a proof situation (for automated proving) with respect to the logical context

2. give explanations on request by learners:
   All underlying mathematics knowledge is transparent due to the “LCF-paradigm” in Isabelle

3. propose a next step if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Demonstration

Summary: Students require these services for learning . . .

1. **check user input** automatically, **flexibly** and reliably:
   Input establishes a *proof situation* (for *automated* proving) with respect to the logical context

2. **give explanations** on request by learners:
   All underlying mathematics knowledge is **transparent** due to the “LCF-paradigm” in Isabelle

3. **propose a next step** if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Students’ View

Demonstration

Summary: Students require these services for learning . . .

1. **check user input** automatically, **flexibly** and reliably:
   Input establishes a *proof situation* (for *automated* proving) with respect to the logical context

2. **give explanations** on request by learners:
   All underlying mathematics knowledge is **transparent** due to the “LCF-paradigm” in Isabelle

3. **propose a next step** if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Demonstration

Summary: Students require these services for learning . . .

1. **check user input** automatically, **flexibly** and reliably:
   - Input establishes a *proof situation* (for *automated* proving)
   - with respect to the logical context

2. **give explanations** on request by learners:
   - All underlying mathematics knowledge is **transparent** due to
   - the “LCF-paradigm” in Isabelle

3. **propose a next step** if learners get stuck:
   - “next-step-guidance” due to Lucas-Interpretation.
Students’ View

Demonstration

Summary: Students require these services for learning . . .

1. check user input automatically, flexibly and reliably:
   Input establishes a *proof situation* (for *automated* proving)
   with respect to the logical context

2. give explanations on request by learners:
   All underlying mathematics knowledge is *transparent* due to
   the “LCF-paradigm” in Isabelle

3. propose a next step if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
Students’ View

Demonstration

Summary: Students require these services for learning . . .

1. **check user input** automatically, **flexibly** and reliably:
   Input establishes a *proof situation* (for *automated* proving) with respect to the logical context

2. **give explanations** on request by learners:
   All underlying mathematics knowledge is **transparent** due to the “LCF-paradigm” in Isabelle

3. **propose a next step** if learners get stuck:
   “next-step-guidance” due to Lucas-Interpretation.
1. User’s Views
   - Demo: Programmers’ View
   - Demo: Students’ View

2. Lucas-Interpretation
   - The Language
   - The Interpreter
   - Where is Interaction from?
   - Summary

3. Conclusions for Users
   - Usability for Programmers
   - Self-explaining System for Students
Another program
with tactics ≈ break-points

. partial_function diffeq_2_mass_oscil (m, l_0, [c_1, c_2],
    d, springs, dampers, sums) =

1   let
11  begin_parallel
1101   springs = Take springs "forces of springs"
111   parallel
1111   dampers = Take dampers "forces of dampers"
112   parallel
1121   sums = Take sums "mass times acceleration equals sum of forces"
12  end_parallel
13  diffeq = Take sums ""
14  diffeq = Substitute [ springs, dampers ]
15  diffeq = Rewrite_Set normalise
16  diffeq = Rewrite_Set vectorify "switch to vector representation"
2   in
21  diffeq
Lucas-Interpretation (LI)
Lucas-Interpretation (LI)

LI: program * location * environment

interpret

location * environment * calculation
Lucas-Interpretation (LI)

LI: program
    * location
    * environment

interpret

location
    * environment
    * calculation

formula or tactic
Lucas-Interpretation (LI)

LI:

- program
  - location
  - environment
- theories
  - context
  - calculation

interpret

location
- environment
- calculation

* context

formula or tactic

Conclusions
Programmers
Students

User’s Views
Programmers
Students

Lucas-Interpretation
Language
Interpreter
Dialogue
Summary

Walther Neuper
Lucas-Interpretation (LI)

LI:

- Program
  - * Location
  - * Environment
  - * Theories
  - * Context
  - * Calculation

* Interpret

Location
  - * Environment
  - * Calculation

* Prove

Context

Formula or Tactic
Lucas-Interpretation (LI)

**Computation**
- program
  - location
  - environment
- LI:
  - theories
  - context
  - calculation
- interpret
  - location
  - environment
  - calculation

**Deduction**
- formula or tactic
Lucas-Interpretation (LI)

* program
  * location
  * environment

LI:

* theories
  * context
  * calculation

interpret

location
  * environment
  * calculation

prove

* context

formula
  or

tactic

computation

semantics of programming languages – settled!

deduction
**Lucas-Interpretation (LI)**

**Computation**  
semantics of programming languages – settled!

**LI:**  
program  
* location  
* environment  
interpret  
location  
* environment  
* calculation

**Deduction**  
semantics of struct.derivations (R.J.Back) – settled!

**LI:**  
* theories  
* context  
* calculation  
prove  
* context

**Formula or Tactic**
Lucas-Interpretation (LI)

**computation**  semantics of programming languages – settled!

LI:

program
  * location
  * environment

interpret

location
  * environment
  * calculation

prove

* theories
  * context
  * calculation

deduction  semantics of struct.derivations (R.J.Back) – settled!
Outline

1. User’s Views
   Demo: Programmers’ View
   Demo: Students’ View

2. Lucas-Interpretation
   The Language
   The Interpreter
   Where is Interaction from?
   Summary

3. Conclusions for Users
   Usability for Programmers
   Self-explaining System for Students
The Dialogue Module

Diagram:

- User Model
- Dialog Rules
- UserAction
- DialogAction
- CalcEvent
- CalcElement
- Worksheet
- Dialogue
- CalcRequest
- CalcEvent
- CalcElement
- Math Engine

Lucas-Interpretation
Walther Neuper
User's Views
Programmers
Students
Lucas-Interpretation
Language
Interpreter
Dialogue
Summary
Conclusions
Programmers
Students
1. User’s Views
   - **Demo**: Programmers’ View
   - **Demo**: Students’ View

2. Lucas-Interpretation
   - The Language
   - The Interpreter
   - Where is Interaction from?
   - **Summary**

3. Conclusions for Users
   - Usability for Programmers
   - Self-explaining System for Students
Lucas-Interpretation is a novel contribution, which

1. interprets a functional language
   - “purely functional” no input / output: interaction $\rightarrow$ Pt.3
   - programmer concerned with mathematics only
   - TODO: embed into Isabelle’s function package

2. controls input / output as side-effects
   - regards tactics as “break points” (like debugger)
   - hands over control at tactics $\rightarrow$ Pt.3

3. delegates user-interaction
to a Dialogue Module:
   - “dialogue authoring” by respective experts
     (DialogRules)
   - adaptive to courses
   - adaptive to individual students UserModel
Lucas-Interpretation is a novel contribution, which

1. interprets a functional language
   - “purely functional” no input / output: interaction $\rightarrow$ Pt.3
   - programmer concerned with mathematics only
   - TODO: embed into Isabelle’s function package

2. controls input / output as side-effects
   - regards tactics as “break points” (like debugger)
   - hands over control at tactics $\rightarrow$ Pt.3

3. delegates user-interaction to a Dialogue Module:
   - “dialogue authoring” by respective experts (DialogRules)
   - adaptive to courses
   - adaptive to individual students UserModel
Summary

Lucas-Interpretation is a novel contribution, which

1. interprets a functional language
   - “purely functional” no input / output: interaction $\rightarrow$ Pt.3
   - programmer concerned with mathematics only
   - TODO: embed into Isabelle’s function package

2. controls input / output as side-effects
   - regards tactics as “break points” (like debugger)
   - hands over control at tactics $\rightarrow$ Pt.3

3. delegates user-interaction to a Dialogue Module:
   - “dialogue authoring” by respective experts (DialogRules)
   - adaptive to courses
   - adaptive to individual students UserModel
1 User’s Views
   Demo: Programmers’ View
   Demo: Students’ View

2 Lucas-Interpretation
   The Language
   The Interpreter
   Where is Interaction from?
   Summary

3 Conclusions for Users
   Usability for Programmers
   Self-explaining System for Students
Usability for Programmers

Programming in \textsc{Isac}

- becomes comparable with Mathematica/Maple/... if \textsc{Isac} adopts Isabelle’s function package
- is embedded into mechanising mathematics, i.e.
  - development of theories (definitions, laws, ...)  
  - development of libraries of specifications  
  - development of verified Computer Algebra
- is separated from users’ interaction:
  interaction is a side-effect managed by Lucas-Interpretation
  - mathematicians focus mathematics
  - interaction is covered by dialogue authors
Outline

1 User’s Views
   **Demo**: Programmers’ View
   **Demo**: Students’ View

2 Lucas-Interpretation
   The Language
   The Interpreter
   Where is Interaction from?
   Summary

3 Conclusions for Users
   Usability for Programmers
   Self-explaining System for Students
Self-explaining system . . .

. . . while step-wise applying a method (solving)

- during trial & error learning:
  - feedback on input steps (formula | tactic)
  - <next> step by system, if got stuck
  - “next-step guidance” by dialogue component:
    - suggest next step partially
    - suggest next steps for selection
    - auto-complete partial input

- in changing levels of abstraction:
  - formal justification for each formula
  - justification = meta-, formula = object-language
  - another “meta-level”: instructions in program
  - . . .

. . . while modelling and specifying an engineering problem: → another talk
Self-explaining system . . .

. . . while step-wise applying a method (solving)

- during trial & error learning:
  - feedback on input steps (formula | tactic)
  - `<next>` step by system, if got stuck
  - “next-step guidance” by dialogue component:
    - suggest next step partially
    - suggest next steps for selection
    - auto-complete partial input

- in changing levels of abstraction:
  - formal *justification* for each *formula*
  - *justification* = meta-, *formula* = object-language
  - another “meta-level”: instructions in program
  - . . .

. . . while *modelling* and *specifying* an engineering problem: → another talk
Thank you for Attention!

F. Haftmann, A. Lochbihler & W. Schreiner.
Towards abstract and executable multivariate polynomials in Isabelle.