GCD — a Case Study on Lucas-Interpretation

Walther Neuper

Institute for Softwaretechnology
Graz University of Technology

Theorem Proving Components for Educational Software (ThEdu)
July 9, 2014
Coimbra, Portugal
1 **Introduction: Lucas-Interpretation (LI)**
   A Concept featuring Step-wise Calculations
   **Demo:** Lucas-Interpretation at Work
   **Summary:** Services for Interactive Learning

2 **Case Study: Which Interactions on GCD of Polynomials?**
   **Demo:** Invariants in Polynomial DIV and GCD
   **Demo:** Step-wise Construction of Polynomial DIV
   **Summary:** two Kinds of Interaction on DIV and GCD

3 **Conclusions for Development**
1 Introduction: Lucas-Interpretation (LI)
A Concept featuring Step-wise Calculations

Demo: Lucas-Interpretation at Work

Summary: Services for Interactive Learning

2 Case Study: Which Interactions on GCD of Polynomials?

Demo: Invariants in Polynomial DIV and GCD
Demo: Step-wise Construction of Polynomial DIV

Summary: two Kinds of Interaction on DIV and GCD

3 Conclusions for Development
Lucas-Interpretation (LI)

\[ I: \text{program} \rightarrow \text{interpret} \rightarrow \text{location} \]

\[ I: \text{location} \rightarrow \text{environment} \]
Lucas-Interpretation (LI)

LI: 
program
* location
* environment

interpret

location
* environment
* calculation
Introduction
Lucas-Interpretation
Demonstration
Services for Learning

Case Study
Invariants in Algorithms
Step-wise Calculation
Kinds of Interaction

Conclusions

Lucas-Interpretation (LI)

LI:
- program
  * location
  * environment

interpret

location
- environment
  * calculation

formula
or
tactic
Lucas-Interpretation (LI)

LI:

- program
  - location
  - environment
- theories
  - context

interpret

location
- environment
- calculation
- context

formula or tactic

Conclusions
Lucas-Interpretation (LI)
Lucas-Interpretation (LI)

**computation**

```
LI:
  * program
    * location
    * environment
  * theories
    * context

interpret
```

**deduction**

```
prove

formula or tactic
```

**location**

```
* environment
  * calculation
  * context
```

**Introduction**
Lucas-Interpretation
Demonstration
Services for Learning

**Case Study**
Invariants in Algorithms
Step-wise Calculation
Kinds of Interaction

**Conclusions**
Lucas-Interpretation (LI)

\[ \text{computation} \quad \text{semantics of programming languages – settled!} \]

**LI:**
- program
  - location
  - environment
- theories
  - context
- interpret
  - location
  - environment
  - calculation
- prove
  - context
- deduction

* formula or tactic
Lucas-Interpretation (LI)

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

**LI:**
- program
  - location
  - environment
- *theories*
  - context

**Interpret**
- location
  - environment
  - calculation

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

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

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

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

**LI:**
- program
  - * location
  - * environment
- * theories
  - * context

**interpret**
- location
  - * environment
  - * calculation

**prove**
- * context

**formula or tactic**

GCD — Lucas-Interpretation
Walther Neuper

Introduction
Lucas-Interpretation
Demonstration
Services for Learning

Case Study
Invariants in Algorithms
Step-wise Calculation
Kinds of Interaction

Conclusions
1 Introduction: Lucas-Interpretation (LI)
   A Concept featuring Step-wise Calculations
   **Demo**: Lucas-Interpretation at Work
   Summary: Services for Interactive Learning

2 Case Study: Which Interactions on GCD of Polynomials?
   **Demo**: Invariants in Polynomial DIV and GCD
   **Demo**: Step-wise Construction of Polynomial DIV
   Summary: two Kinds of Interaction on DIV and GCD

3 Conclusions for Development
1. Introduction: Lucas-Interpretation (LI)
   A Concept featuring Step-wise Calculations
   Demo: Lucas-Interpretation at Work
   Summary: Services for Interactive Learning

2. Case Study: Which Interactions on GCD of Polynomials?
   Demo: Invariants in Polynomial DIV and GCD
   Demo: Step-wise Construction of Polynomial DIV
   Summary: two Kinds of Interaction on DIV and GCD

3. Conclusions for Development
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.

The services support . . .

- **step-wise solving math** problems in STEM
- learning interactively like with a **chess-program**

---

1“STEM” is Science, Technology, Engineering and Mathematics
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.

The services support . . .

- **step-wise solving math** problems in STEM
- learning interactively like with a **chess-program**

---

1 “STEM” is Science, Technology, Engineering and Mathematics.
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.

The services support . . .

- **step-wise solving math** problems in STEM ¹
- learning interactively like with a **chess-program**

¹“STEM” is Science, Technology, Engineering and Mathematics
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.

The services support . . .

- **step-wise solving math** problems in STEM \(^1\)
- learning interactively like with a **chess-program**

---

\(^1\)“STEM” is Science, Technology, Engineering and Mathematics.
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.

The services support...

- step-wise solving math problems in STEM ¹
- learning interactively like with a **chess-program**

---

¹“STEM” is Science, Technology, Engineering and Mathematics.
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.

The services support . . .

- **step-wise solving math** problems in STEM
- learning interactively like with a **chess-program**

---

1 “STEM” is **Science, Technology, Engineering and Mathematics**
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.

The services support . . .

- **step-wise solving math** problems in STEM

- learning interactively like with a **chess-program**

---

1 “STEM” is **Science, Technology, Engineering and Mathematics**
check user input automatically, *flexibly* and reliably:
Input establishes a *proof situation* (for *automated* proving)
with respect to the logical context

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

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

The services support . . .

- **step-wise solving math** problems in STEM

---

1“**STEM**” is **Science, Technology, Engineering and Mathematics**.
check user input automatically, flexibly and reliably:
Input establishes a proof situation (for automated proving) with respect to the logical context

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

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

The services support ...

- step-wise solving math problems in STEM ¹
- learning interactively like with a chess-program

¹“STEM” is Science, Technology, Engineering and Mathematics.
1 Introduction: Lucas-Interpretation (LI)
A Concept featuring Step-wise Calculations
Demo: Lucas-Interpretation at Work
Summary: Services for Interactive Learning

2 Case Study: Which Interactions on GCD of Polynomials?
Demo: Invariants in Polynomial DIV and GCD
Demo: Step-wise Construction of Polynomial DIV
Summary: two Kinds of Interaction on DIV and GCD

3 Conclusions for Development
Introduction: Lucas-Interpretation (LI)
A Concept featuring Step-wise Calculations
Demo: Lucas-Interpretation at Work
Summary: Services for Interactive Learning

Case Study: Which Interactions on GCD of Polynomials?
Demo: Invariants in Polynomial DIV and GCD
Demo: Step-wise Construction of Polynomial DIV
Summary: two Kinds of Interaction on DIV and GCD

Conclusions for Development
Outline

1. Introduction: Lucas-Interpretation (LI)
   A Concept featuring Step-wise Calculations
   **Demo:** Lucas-Interpretation at Work
   **Summary:** Services for Interactive Learning

2. Case Study: Which Interactions on GCD of Polynomials?
   **Demo:** Invariants in Polynomial DIV and GCD
   **Demo:** Step-wise Construction of Polynomial DIV
   **Summary:** two Kinds of Interaction on DIV and GCD

3. Conclusions for Development
Two Kinds of Interaction

1. Learning by observation of invariants
   - Invariants capture the essence of algorithms
   - Such essence is not immediately evident
   - A “notepad” is required for specific investigations; it should “know” the elements of a calculation!

2. Learning by step-wise construction of solutions
   - Trial-and-error is an indispensable principle for learning
   - Isac needs flexible mechanisms for structured in/output
   - We need adaptive switching between
     - input by student and check/feedback by system
     - auto-completion by the system (triggered by a dialog module, not only a next-button)
Two Kinds of Interaction

1 Learning by observation of invariants
   - Invariants capture the essence of algorithms
   - Such essence is not immediately evident
   - A “notepad” is required for specific investigations; it should “know” the elements of a calculation!

2 Learning by step-wise construction of solutions
   - Trial-and-error is an indispensable principle for learning
   - Isac needs flexible mechanisms for structured in/output
   - We need adaptive switching between
     - input by student and check/feedback by system
     - auto-completion by the system (triggered by a dialog module, not only a next-button)
Conclusions for Development

1. Implement GCD by pseudo-division i.e. *no* traditional modular approach, *no* residues.
2. Extend Lucas-Interpreter to Isabelle functions, re-use mechanisms of Isabelle’s code generator.
3. Extend the prog.language with tactic INVARIANT, a breakpoint in Lucas-Interpreter.
4. Design structured in/output (e.g. step-wise DIV) with several i/o steps at one step of Lucas-Interpreter and a clean interface to representation at the frontend.
5. Exploit interaction enabled by the new services (Lucas-Interpreter (SML) separated from frontend (Java))
Conclusions for Development

1. Implement GCD by pseudo-division i.e. *no* traditional modular approach, *no* residues.

2. Extend Lucas-Interpreter to Isabelle functions, re-use mechanisms of Isabelle’s code generator.

3. Extend the prog.language with tactic `INARIANT`, a breakpoint in Lucas-Interpreter.

4. Design structured in/output (e.g. step-wise `DIV`) with several i/o steps at one step of Lucas-Interpreter and a clean interface to representation at the frontend.

5. Exploit interaction enabled by the new services (Lucas-Interpreter (SML) separated from frontend (Java))
Conclusions for Development

1. Implement GCD by pseudo-division i.e. *no* traditional modular approach, *no* residues.
2. Extend Lucas-Interpreter to Isabelle functions, re-use mechanisms of Isabelle’s code generator.
3. Extend the prog.language with tactic `INARIANT`, a breakpoint in Lucas-Interpreter.
4. Design structured in/output (e.g. step-wise `DIV`) with several i/o steps at one step of Lucas-Interpreter and a clean interface to representation at the frontend.
5. Exploit interaction enabled by the new services (Lucas-Interpreter (SML) separated from frontend (Java))
Conclusions for Development

1. Implement GCD by pseudo-division i.e. *no* traditional modular approach, *no* residues.

2. Extend Lucas-Interpreter to Isabelle functions, re-use mechanisms of Isabelle’s code generator.

3. Extend the prog.language with tactic `INARIANT`, a breakpoint in Lucas-Interpreter.

4. Design structured in/output (e.g. step-wise `DIV`) with several i/o steps at one step of Lucas-Interpreter and a clean interface to representation at the frontend.

5. Exploit interaction enabled by the new services (Lucas-Interpreter (SML) separated from frontend (Java))
Conclusions for Development

1. Implement GCD by pseudo-division i.e. *no* traditional modular approach, *no* residues.
2. Extend Lucas-Interpreter to Isabelle functions, re-use mechanisms of Isabelle’s code generator.
3. Extend the prog.language with tactic `INVARIANT`, a breakpoint in Lucas-Interpreter.
4. Design structured in/output (e.g. step-wise `DIV`) with several i/o steps at one step of Lucas-Interpreter and a clean interface to representation at the frontend.
5. Exploit interaction enabled by the new services (Lucas-Interpreter (SML) separated from frontend (Java))
Thank you for Attention!