Master Theses: *Implementation of Computer Algebra in the Theorem Prover* Isabelle

Computation meets Deduction

Wolfgang Schreiner
Walther Neuper

Research Institute for Symbolic Computation, JKU Linz,
Institute for Software Technology, TU Graz

Linz, 12.4.2013

Outline

Outline

Promote "Formal Methods" (+ Róle of Mathematicians)

- "Formal Methods" (FM) — an advancing discipline: http://www.fmeurope.org/?page_id=2
  - Conferences on FM: http://lipn.univ-paris13.fr/~andre/conferences.php
    - 11th International Conference on Software Engineering and Formal Methods http://antares.sip.ucm.es/sefm2013/
    - FM-RAIL-BOK WORKSHOP 2013 http://ssfmgroup.wordpress.com/about/
    - iFM 2013: 10th International Conference on integrated Formal Methods http://www.it.abo.fi/iFM2013/
    - ...
  - Applications of FM:
    - railway operation and control systems http://www.informatik.uni-bremen.de/agbs/lehre/tracs/
    - circuit design ("Pentium Bug") http://www.csl.sri.com/papers/computer96/computer96.html
    - robotics http://cps-vo.org/node/1134
    - ...
- promotes róle for mathematicians in engineering by FM:

Promote "Formal Methods" (+ Róle of Mathematicians)

- "Formal Methods" (FM) — an advancing discipline: `http://www.fmeurope.org/?page_id=2`
  - Conferences on FM: `http://lipn.univ-paris13.fr/~andre/conferences.php`
    - 11th International Conference on Software Engineering and Formal Methods `http://antares.sip.ucm.es/sefm2013/`
    - FM-RAIL-BOK WORKSHOP 2013 `http://ssfmgroup.wordpress.com/about/`
    - iFM 2013: 10th International Conference on integrated Formal Methods `http://www.it.abo.fi/iFM2013/`
    - ...
  - Applications of FM:
    - railway operation and control systems `http://www.informatik.uni-bremen.de/agbs/lehre/tracs/`
    - circuit design ("Pentium Bug") `http://www.csl.sri.com/papers/computer96/computer96.html`
    - robotics `http://cps-vo.org/node/1134`
    - ...
- promotes róle for mathematicians in engineering by FM:

Promote "Formal Methods" (+ Róle of Mathematicians)

- "Formal Methods" (FM) — an advancing discipline:
- promotes róle for mathematicians in engineering:
  - FM extends the field of application of mathematics

```
definition segment :: "Point ⇒ Point ⇒ Point set"
where "segment x y =
  {z. ∃t . 0≤t ∧ t≤1 ∧ (z = t *R x + (1-t) *R y) }"

definition is_convex :: "Point set ⇒ bool"
where "is_convex K = (∀x∈K. ∀y∈K. segment x y ⊆ K)"
```

  - FM raises demand for specifying systems' features
  - FM raise demand for verified implementation
  - ... thus increases involvement of mathematicians.

Promote “Formal Methods” (+ Róle of Mathematicians)

- “Formal Methods” (FM) — an advancing discipline:
- promotes róle for mathematicians in engineering:
  - FM extends the field of application of mathematics
  - FM raises demand for specifying systems’ features

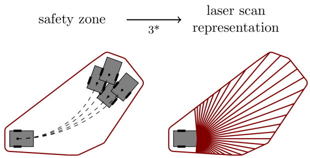


Fig. 5 Postprocessing step 3: Convert safety zone from internal Sphere Swept Convex Hull (SSCH) representation into a laser scan like representation. In this representation the safety zone can be safeguarded by simply comparing with a laser scan.

  - FM raise demand for verified implementation
  - . . . thus increases involvement of mathematicians.

Promote “Formal Methods” (+ Róle of Mathematicians)

- “Formal Methods” (FM) — an advancing discipline:
- promotes róle for mathematicians in engineering:
  - FM extends the field of application of mathematics
  - FM raises demand for specifying systems’ features
  - FM raise demand for verified implementation

$T(s,\alpha)$. This results in the overall computation

$$H(s_{\min}, s_{\max}, \alpha_{\min}, \alpha_{\max}) = A\left(\left[\left[\left[P^1_{i,s,\alpha}, P^2_{i,s,\alpha}, [V^j_{i,s,\alpha}]_{j=0}^{L-1}\right]_{i=0}^{n}\right]_{s_{\min}, s_{\max}}\right]_{\alpha_{\min}, \alpha_{\max}} ; q\right),$$

with

$$q = q^A + q^B$$
$$q^A = \frac{1}{6}\left(\frac{\alpha_{\max}-\alpha_{\min}}{2}\right)^2 \max\{|s_{\max}|; |s_{\min}|\}$$
$$q^B = \left(1 - \cos\frac{\alpha_{\max}-\alpha_{\min}}{2}\right) \max_{1\le i\le n}\{|R_i|\}.$$

- …thus increases involvement of mathematicians.

Promote “Formal Methods” (+ Róle of Mathematicians)

- “Formal Methods” (FM) — an advancing discipline:
- promotes róle for mathematicians in engineering:
  - FM extends the field of application of mathematics
  - FM raises demand for specifying systems’ features
  - FM raise demand for verified implementation
  - . . . thus increases involvement of mathematicians.

Fig. 1 The SAMS demonstrator driving a right hand bent and the collision-free safety zone of that movement. If there was any obstacle inside the safety zone the AGV would stop.

Outline

Develop
Verified Computer Algebra

- FM require essential functionality of systems **verified**
- functionality is determined by (hard- and) **software** components
- Computer Algebra **(CA) is foundamental** for software
- so **FM requires verified implementation of CA**.

We shall implement selected CA algorithms in the Theorem Prover Isabelle —

— *using* Isabelle*'s recent "function package" ("computation"), which automates much of verification tasks ("deduction").*

Develop Verified Computer Algebra

- FM require essential functionality of systems **verified**
- functionality is determined by (hard- and) **software** components
- Computer Algebra **(CA) is foundamental** for software
- so **FM requires verified implementation of CA**.

We shall implement selected CA algorithms in the Theorem Prover Isabelle —

— *using* Isabelle*'s recent "function package" ("computation"), which automates much of verification tasks ("deduction").*

Develop Verified Computer Algebra

- FM require essential functionality of systems **verified**
- functionality is determined by (hard- and) **software** components
- Computer Algebra **(CA) is foundamental** for software
- so **FM requires verified implementation of CA.**

We shall implement selected CA algorithms in the Theorem Prover Isabelle —

— *using* Isabelle*'s recent "function package" ("computation"), which automates much of verification tasks ("deduction").*

Develop Verified Computer Algebra

- FM require essential functionality of systems **verified**
- functionality is determined by (hard- and) **software** components
- Computer Algebra **(CA) is foundamental** for software
- so **FM requires verified implementation of CA**.

We shall implement selected CA algorithms in the Theorem Prover Isabelle —

— *using* Isabelle*'s recent "function package" ("computation"), which automates much of verification tasks ("deduction").*

Develop Verified Computer Algebra

- FM require essential functionality of systems **verified**
- functionality is determined by (hard- and) **software** components
- Computer Algebra **(CA) is foundamental** for software
- so **FM requires verified implementation of CA**.

We shall implement selected CA algorithms in the Theorem Prover Isabelle —

— *using* Isabelle*'s recent "function package" ("computation"), which automates much of verification tasks ("deduction").*

Outline

"Transparent" Systems for Education

- Learning math starts with **algorithms**: simplify, gcd, ...
- **Understanding** an algorithm requires both,

program and	**specification**
fun $gcd :: nat \Rightarrow nat \Rightarrow nat$ where	$gcd :: nat \Rightarrow nat \Rightarrow nat$
"$gcd\ a\ 0 = a$"	$gcd\ a\ b = c$
| "$gcd\ a\ b = if\ a < b$	assumes $a \neq 0$
$then\ gcd\ a\ (b\ mod\ a)$	yields $c\ dvd\ a \wedge c\ dvd\ b \wedge$
$else\ gcd\ b\ (a\ mod\ b)$"	$\forall c'.(c'\ dvd\ a \wedge c'\ dvd\ b) \Rightarrow c' \leq c$

- $\mathcal{ISAC}$ "explains itself"; so it is **transparent** to both
- and combines computation and deduction by "**Lucas-Interpretation**":

computation and	**deduction**
environment: $\epsilon = \{(a, 75), (b, 6), \ldots\}$	logical context: $c = \{a \neq 0, a \geq b \Rightarrow \ldots$

We re-use CA algorithms implemented in Isabelle *in the experimental* $\mathcal{ISAC}$ *system (which is based on* Isabelle*) —*

— $\mathcal{ISAC}$*, a "transparent system for learning mathematics"*

"Transparent" Systems for Education

- Learning math starts with **algorithms**: simplify, gcd, …
- **Understanding** an algorithm requires both,

program	and	**specification**
fun $gcd :: nat \Rightarrow nat \Rightarrow nat$ where		$gcd :: nat \Rightarrow nat \Rightarrow nat$
"$gcd\ a\ 0 = a$"		$gcd\ a\ b = c$
| "$gcd\ a\ b = if\ a < b$		assumes $a \neq 0$
$then\ gcd\ a\ (b\ mod\ a)$		yields $c\ dvd\ a \wedge c\ dvd\ b \wedge$
$else\ gcd\ b\ (a\ mod\ b))$"		$\forall c'.(c'\ dvd\ a \wedge c'\ dvd\ b) \Rightarrow c' \leq c$

- $\mathcal{ISAC}$ "explains itself"; so it is **transparent** to both
- and combines computation and deduction by "**Lucas-Interpretation**":

computation	and	**deduction**
environment: $\epsilon = \{(a, 75), (b, 6), \ldots\}$		logical context: $c = \{a \neq 0, a \geq b \Rightarrow \ldots$

We re-use CA algorithms implemented in Isabelle
in the experimental $\mathcal{ISAC}$ *system (which is based on* Isabelle*) —*

— $\mathcal{ISAC}$*, a "transparent system for learning mathematics"*

"Transparent" Systems for Education

- Learning math starts with **algorithms**: simplify, gcd, ...
- **Understanding** an algorithm requires both,

program and

fun *gcd* :: *nat* ⇒ *nat* ⇒ *nat* where
  "*gcd a 0 = a*"
| "*gcd a b = if a* < *b*
    then gcd a (b mod a)
    else gcd b (a mod b))"

specification

gcd :: *nat* ⇒ *nat* ⇒ *nat*
gcd a b = c
  assumes $a \neq 0$
  yields c *dvd* $a \wedge c$ *dvd* $b \wedge$
    $\forall c'.(c'$ *dvd* $a \wedge c'$ *dvd* $b) \Rightarrow c' \leq c$

- $\mathcal{ISAC}$ "explains itself"; so it is **transparent** to both
- and combines computation and deduction by "**Lucas-Interpretation**":

computation and

environment: $\epsilon = \{(a, 75), (b, 6), \ldots\}$

deduction

logical context: $c = \{a \neq 0, a \geq b \Rightarrow \ldots$

We re-use CA algorithms implemented in Isabelle
in the experimental $\mathcal{ISAC}$ *system (which is based on* Isabelle*)* —

— $\mathcal{ISAC}$*, a "transparent system for learning mathematics"*

"Transparent" Systems for Education

- Learning math starts with **algorithms**: simplify, gcd, ...
- **Understanding** an algorithm requires both,

program	and	**specification**
fun *gcd* :: *nat* ⇒ *nat* ⇒ *nat* where "*gcd a 0 = a*" | "*gcd a b = if a < b* *then gcd a (b mod a)* *else gcd b (a mod b))*"		*gcd* :: *nat* ⇒ *nat* ⇒ *nat* *gcd a b = c* assumes $a \neq 0$ yields *c dvd a* ∧ *c dvd b* ∧ ∀*c′*.(*c′ dvd a* ∧ *c′ dvd b*) ⇒ $c' \leq c$

- $\mathcal{ISAC}$ "explains itself"; so it is **transparent** to both
- and combines computation and deduction by "**Lucas-Interpretation**":

computation	and	**deduction**
environment: $\epsilon = \{(a, 75), (b, 6), \ldots\}$		logical context: $c = \{a \neq 0, a \geq b \Rightarrow \ldots$

We re-use CA algorithms implemented in Isabelle
in the experimental $\mathcal{ISAC}$ system (which is based on Isabelle*)* —

— *$\mathcal{ISAC}$, a "transparent system for learning mathematics"*

"Transparent" Systems for Education

- Learning math starts with **algorithms**: simplify, gcd, . . .
- **Understanding** an algorithm requires both,

program	and	**specification**
fun *gcd* :: *nat* ⇒ *nat* ⇒ *nat* where		*gcd* :: *nat* ⇒ *nat* ⇒ *nat*
"*gcd a 0 = a*"		*gcd a b = c*
| "*gcd a b = if a* $<$ *b*		assumes $a \neq 0$
then gcd a (b mod a)		yields c *dvd* $a \wedge c$ *dvd* $b \wedge$
else gcd b (a mod b))"		$\forall c'.(c'$ *dvd* $a \wedge c'$ *dvd* $b) \Rightarrow c' \leq c$

- $\mathcal{ISAC}$ "explains itself"; so it is **transparent** to both
- and combines computation and deduction by "**Lucas-Interpretation**":

computation	and	**deduction**
environment: $\epsilon = \{(a, 75), (b, 6), \ldots\}$		logical context: $c = \{a \neq 0, a \geq b \Rightarrow \ldots$

We re-use CA algorithms implemented in Isabelle
in the experimental $\mathcal{ISAC}$ *system (which is based on* Isabelle*)* —

— $\mathcal{ISAC}$*, a "transparent system for learning mathematics"*

$\mathcal{ISAC}$-Project Graz

- $\mathcal{ISAC}$ development started 1997 under supervision by Peter Lucas, Graz and Bruno Buchberger, Linz. `http://www.ist.tugraz.at/isac/`
- $\mathcal{ISAC}$ is an experimental system for math education
  - $\mathcal{ISAC}$ is "$\mathcal{ISA}$belle for $\mathcal{C}$alculations in applied math"
  - math-engine builds upon Isabelle
  - "Lucas-Interpretation"
    1. determines a next step in a calculation
    2. checks a step (expression / thm) input by students
- Development at Graz focuses user interfaces and dialogs,
- and requires Computer Algebra (CA) which is not yet implemented in Isabelle

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

$\mathcal{ISAC}$-Project Graz

- $\mathcal{ISAC}$ development started 1997 under supervision by Peter Lucas, Graz and Bruno Buchberger, Linz.
  `http://www.ist.tugraz.at/isac/`
- $\mathcal{ISAC}$ is an experimental system for math education
  - $\mathcal{ISAC}$ is "$\mathcal{ISA}$belle for $\mathcal{C}$alculations in applied math"
  - math-engine builds upon Isabelle
  - "Lucas-Interpretation"
    1. determines a next step in a calculation
    2. checks a step (expression / thm) input by students
- Development at Graz focuses user interfaces and dialogs,
- and requires Computer Algebra (CA) which is not yet implemented in Isabelle

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

$\mathcal{ISAC}$-Project Graz

- $\mathcal{ISAC}$ development started 1997 under supervision by Peter Lucas, Graz and Bruno Buchberger, Linz.
  `http://www.ist.tugraz.at/isac/`
- $\mathcal{ISAC}$ is an experimental system for math education
  - $\mathcal{ISAC}$ is "$\mathcal{ISA}$belle for $\mathcal{C}$alculations in applied math"
  - math-engine builds upon Isabelle
  - "Lucas-Interpretation"
    1. determines a next step in a calculation
    2. checks a step (expression / thm) input by students
- Development at Graz focuses user interfaces and dialogs,
- and requires Computer Algebra (CA)
  which is not yet implemented in Isabelle

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

$\mathcal{ISAC}$-Project Graz

- $\mathcal{ISAC}$ development started 1997 under supervision by Peter Lucas, Graz and Bruno Buchberger, Linz.
  `http://www.ist.tugraz.at/isac/`
- $\mathcal{ISAC}$ is an experimental system for math education
  - $\mathcal{ISAC}$ is "$\mathcal{ISA}$belle for $\mathcal{C}$alculations in applied math"
  - math-engine builds upon Isabelle
  - "Lucas-Interpretation"
    1. determines a next step in a calculation
    2. checks a step (expression / thm) input by students
- Development at Graz focuses user interfaces and dialogs,
- and requires Computer Algebra (CA) which is not yet implemented in Isabelle

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

$\mathcal{ISAC}$-Project Graz

- $\mathcal{ISAC}$ development started 1997 under supervision by Peter Lucas, Graz and Bruno Buchberger, Linz.
  `http://www.ist.tugraz.at/isac/`
- $\mathcal{ISAC}$ is an experimental system for math education
  - $\mathcal{ISAC}$ is "$\mathcal{ISA}$belle for $\mathcal{C}$alculations in applied math"
  - math-engine builds upon Isabelle
  - "Lucas-Interpretation"
    1. determines a next step in a calculation
    2. checks a step (expression / thm) input by students
- Development at Graz focuses user interfaces and dialogs,
- and requires Computer Algebra (CA)
  which is not yet implemented in Isabelle

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

Outline

How the Simplifier Works

We demonstrate “life in Isabelle” how to

- rewrite by ’rule’ and ’by simp’lification
- use the simplifier as a proof tool
- evaluate functions using the simplifier
- investigate Isabelle’s “transparent” knowledge.

Isabelle’s simplifier is

- a general and powerful proof tool
- frequently usedin Isabelle proofs
- highly efficient on large “simp-sets” due to “discrimination-nets”

How the Simplifier Works

We demonstrate “life in Isabelle” how to

- rewrite by ’rule’ and ’by simp’lification
- use the simplifier as a proof tool
- evaluate functions using the simplifier
- investigate Isabelle’s “transparent” knowledge.

Isabelle’s simplifier is

- a general and powerful proof tool
- frequently usedin Isabelle proofs
- highly efficient on large “simp-sets” due to “discrimination-nets”

Outline

Logical Foundations

- Equations are *proved* theorems (rewrite rules):
  - no extraneous variables in right-hand sides
  - left-hand sides are “higher-order patterns” (+functions)
- “Bottom-up” rewriting after preprocessing:

$$\begin{array}{rcl} \neg P & \mapsto & P = \mathrm{False} \\ P \longrightarrow Q & \mapsto & P \Longrightarrow Q \\ P \wedge Q & \mapsto & P \; , \; Q \\ \forall x.\ P\ x & \mapsto & P\ ?x \\ \forall x \in A.\ P\ x & \mapsto & ?x \in A \Longrightarrow P\ ?x \\ \mathrm{if}\ P\ \mathrm{then}\ Q\ \mathrm{else}\ R & \mapsto & P \Longrightarrow Q,\ \neg P \Longrightarrow R \\ \textit{remaining non-equations}\ P & \mapsto & P = \mathrm{True} \end{array}$$

  ($P \Longrightarrow Q$ creates a “context” P for conditional rewriting of Q)
- Conditional rewriting, ordered rewriting (lexicographic order)
- Congruence rules for $\longrightarrow, \forall, \exists$, *if..then..else*, etc:
  e.g. for $\longrightarrow$: $[[P = P'; P' \Longrightarrow Q = Q']] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Logical Foundations

- Equations are *proved* theorems (rewrite rules):
  - no extraneous variables in right-hand sides
  - left-hand sides are "higher-order patterns" (+functions)
- "Bottom-up" rewriting after preprocessing:

$$\begin{array}{rcl}
\neg P & \mapsto & P = \mathrm{False} \\
P \longrightarrow Q & \mapsto & P \Longrightarrow Q \\
P \wedge Q & \mapsto & P\ ,\ Q \\
\forall x.\ P\ x & \mapsto & P\ ?x \\
\forall x \in A.\ P\ x & \mapsto & ?x \in A \Longrightarrow P\ ?x \\
\mathrm{if}\ P\ \mathrm{then}\ Q\ \mathrm{else}\ R & \mapsto & P \Longrightarrow Q,\ \neg P \Longrightarrow R \\
\mathit{remaining\ non{-}equations}\ P & \mapsto & P = \mathrm{True}
\end{array}$$

($P \Longrightarrow Q$ creates a "context" P for conditional rewriting of Q)

- Conditional rewriting, ordered rewriting (lexicographic order)
- Congruence rules for $\longrightarrow, \forall, \exists$, *if..then..else*, etc:
  e.g. for $\longrightarrow$: $[\![P = P'; P' \Longrightarrow Q = Q']\!] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Logical Foundations

- Equations are *proved* theorems (rewrite rules):
  - no extraneous variables in right-hand sides
  - left-hand sides are “higher-order patterns” (+functions)
- “Bottom-up” rewriting after preprocessing:

$$
\begin{array}{rcl}
\neg P & \mapsto & P = \mathrm{False} \\
P \longrightarrow Q & \mapsto & P \Longrightarrow Q \\
P \wedge Q & \mapsto & P\ ,\ Q \\
\forall x.\ P\ x & \mapsto & P\ ?x \\
\forall x \in A.\ P\ x & \mapsto & ?x \in A \Longrightarrow P\ ?x \\
\mathrm{if}\ P\ \mathrm{then}\ Q\ \mathrm{else}\ R & \mapsto & P \Longrightarrow Q,\ \neg P \Longrightarrow R \\
\textit{remaining non-equations}\ P & \mapsto & P = \mathrm{True}
\end{array}
$$

($P \Longrightarrow Q$ creates a “context” P for conditional rewriting of Q)

- Conditional rewriting, ordered rewriting (lexicographic order)
- Congruence rules for $\longrightarrow, \forall, \exists$, *if..then..else*, etc:
  e.g. for $\longrightarrow$: $[[P = P'; P' \Longrightarrow Q = Q']] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Logical Foundations

- Equations are *proved* theorems (rewrite rules):
  - no extraneous variables in right-hand sides
  - left-hand sides are "higher-order patterns" (+functions)
- "Bottom-up" rewriting after preprocessing:

$$\begin{array}{rcl}
\neg P & \mapsto & P = \mathrm{False} \\
P \longrightarrow Q & \mapsto & P \Longrightarrow Q \\
P \wedge Q & \mapsto & P\ ,\ Q \\
\forall x.\ P\ x & \mapsto & P\ ?x \\
\forall x \in A.\ P\ x & \mapsto & ?x \in A \Longrightarrow P\ ?x \\
\mathrm{if}\ P\ \mathrm{then}\ Q\ \mathrm{else}\ R & \mapsto & P \Longrightarrow Q,\ \neg P \Longrightarrow R \\
\mathit{remaining\ non{-}equations}\ P & \mapsto & P = \mathrm{True}
\end{array}$$

  ($P \Longrightarrow Q$ creates a "context" P for conditional rewriting of Q)
- Conditional rewriting, ordered rewriting (lexicographic order)
- Congruence rules for $\longrightarrow, \forall, \exists$, *if..then..else*, etc:
  e.g. for $\longrightarrow$: $[[P = P'; P' \Longrightarrow Q = Q']] \Longrightarrow (P \longrightarrow Q) = (P' \longrightarrow Q')$

Outline

GCD Algorithm for Polynomials

Thesis Description:
Implement and Verify a GCD Algorithm for Polynomials in Isabelle
`http://www.risc.jku.at/education/theses/?view=53`

Demonstration:
ML code: `~~/src/Tools/isac/Knowledge/GCD_Poly.thy`
Translation to Isabelle:
`~~/src/Tools/isac/Knowledge/GCD_Poly_FP.thy`
Translated code:
`~~/test/Tools/isac/Knowledge/gcd_poly.sml`

Outline

1. Motivations and Goals
   - Promote "Formal Methods" (+ Róle of Mathematicians)
   - Develop Verified Computer Algebra
   - Make Systems "Transparent" for Education
2. Interlude: Isabelle's Simplifier
   - How the Simplifier Works
   - Logical Foundations of the Simplifier
3. Topics for Master Theses
   - GCD Algorithm for Polynomials
   - "Groebner_Basis.thy" for Equation Solving
   - "Multivalued Functions" in Simplification
4. Benefits for Students
5. Download:
   http://www.ist.tugraz.at/projects/isac/www/download/RISC_Theses_presentation.pd

"Groebner_Basis.thy" for Equation Solving

Thesis Description:
Promote Isabelle's "Groebner.thy" to Equation Solving
`http://www.risc.jku.at/education/theses/?view=54`

Download Isabelle:
`http://isabelle.in.tum.de/index.html`
Isabelle NEWs:
`http://www21.in.tum.de/`
World map of Isabelle users:
`http://isabelle.in.tum.de/google_map.html`

Demonstration:
Example: `~~/src/HOL/ex/Groebner_Examples.thy`
"Transparent" knowledge: `~~/src/HOL/Rings.thy`
Theory: `~~/src/HOL/Groebner_Basis.thy`
Translate code: `~~/src/HOL/Tools/groebner.ML`

Outline

“Multivalued Functions” in Simplification

Thesis Description:
“Multivalued Functions” in Reliable Algebraic Simplification
`http://www.risc.jku.at/education/theses/?view=55`

Demonstration:
sin, cos, tan, arcsin, DERIVative, power series: `http://isabelle.in.tum.de/dist/library/HOL/Transcendental.html`
log, DERIVative, power series:
`http://isabelle.in.tum.de/dist/library/HOL/Ln.html`

“multivalued functions”

Examples: terms with multivalued “functions”:

$$\sin x = y$$

$$x = \arcsin y \quad ???$$

$$x = \begin{cases} \{\} & y < -1 \;\vee\; 1 < y \\ \{x.\ \sin x = y \wedge \forall x'.\ \sin x' = y \Rightarrow \\ \quad \exists k \in \mathcal{Z}.\ x' = 2k\pi - x \vee x' = (2k+1)\pi + x\} & -1 \leq y \;\wedge\; y < 0 \\ \{x.\ \sin x = y \wedge \forall x'.\ \sin x' = y \Rightarrow \\ \quad \exists k \in \mathcal{Z}.\ x' = 2k\pi + x \vee x' = (2k+1)\pi - x\} & 0 \leq y \;\wedge\; y \leq 1 \end{cases}$$

$$\arctan x + \arctan y = \mathit{arctan}\left(\frac{x+y}{1-xy}\right) + \begin{cases} \pi & xy > 1 \wedge x > 0 \\ 0 & xy < 1 \\ -\pi & xy > 1 \wedge x < 0 \end{cases}$$

These terms give raise to **branching problems**, branches connected with assumptions.

“multivalued functions”

Major deficiencies in state-of-the-art CA result from (1) under-*specification* and (2) weak *deductive mechanisms*.

Isabelle *has* concepts and mechanisms to tackle the deficiencies:

1. log, exp, sin, arcsin, *etc* are rigorously *specified* as functions (not relations)
2. *Deductive mechanisms* are present: provers, contexts, tactics, etc.

Using these mechanisms promise substantial advances in:

- Simplification involving multivalued “functions”, radicals.
- Integration of terms containing multivalued “functions”
- Equation solving with multivalued “functions”.

“multivalued functions”

Major deficiencies in state-of-the-art CA result from (1) under-*specification* and (2) weak *deductive mechanisms*.

Isabelle *has* concepts and mechanisms to tackle the deficiencies:

1. log, exp, sin, arcsin, *etc* are rigorously *specified* as functions (not relations)
2. *Deductive mechanisms* are present: provers, contexts, tactics, etc.

Using these mechanisms promise substantial advances in:

- Simplification involving multivalued “functions”, radicals.
- Integration of terms containing multivalued “functions”
- Equation solving with multivalued “functions”.

“multivalued functions”

Possible directions:

- Clarify principles for overcoming deficiencies in CA
  - Which provers for which kinds of assumptions ?
  - Handle assumptions using Isabelle’s contexts
  - Expressiveness of assumptions (full predicate calculus ?)
- Design specific improvements for particular deficiencies
- Implement improvements for selected topics in Isabelle

Open issues:

- Interplay between computation and deduction during simplification: How stop simplifier for
  - for automated proof ?
  - for interactive proving ?
  - for debugging (inspecting context, etc) ?
- Code generation for functions with integrated proving
- How call deductive mechanisms from generated code ?

“multivalued functions”

Possible directions:

- Clarify principles for overcoming deficiencies in CA
  - Which provers for which kinds of assumptions ?
  - Handle assumptions using Isabelle’s contexts
  - Expressiveness of assumptions (full predicate calculus ?)
- Design specific improvements for particular deficiencies
- Implement improvements for selected topics in Isabelle

Open issues:

- Interplay between computation and deduction during simplification: How stop simplifier for
  - for automated proof ?
  - for interactive proving ?
  - for debugging (inspecting context, etc) ?
- Code generation for functions with integrated proving
- How call deductive mechanisms from generated code ?

"multivalued functions"

Summary:

"Multivalued functions" is an important topic, suffers from long-standing deficiencies unresolved in CA.

Handling assumptions appears most promising by deductive mechanisms . . . not yet tackled, because

- *CA developers are not interested (e.g. no logics) ?*
- *TP systems are not ready (e.g. no complex functions yet) ?*

Calling deductive mechanisms during computation would require Isabelle to adapt

- *the simplifier for calls during "execution" of functions*
- *the code generator for inserting "call-backs" to provers.*

“multivalued functions”

Summary:

“Multivalued functions” is an important topic, suffers from long-standing deficiencies unresolved in CA.

Handling assumptions appears most promising by deductive mechanisms . . . not yet tackled, because

- *CA developers are not interested (e.g. no logics) ?*
- *TP systems are not ready (e.g. no complex functions yet) ?*

Calling deductive mechanisms during computation would require Isabelle to adapt

- *the simplifier for calls during “execution” of functions*
- *the code generator for inserting “call-backs” to provers.*

"multivalued functions"

Summary:

"Multivalued functions" is an important topic, suffers from long-standing deficiencies unresolved in CA.

Handling assumptions appears most promising by deductive mechanisms ... not yet tackled, because

- *CA developers are not interested (e.g. no logics) ?*
- *TP systems are not ready (e.g. no complex functions yet) ?*

Calling deductive mechanisms during computation would require Isabelle to adapt

- *the simplifier for calls during "execution" of functions*
- *the code generator for inserting "call-backs" to provers.*

Benefits for Students

- **Get a thorough introduction to Isabelle**
  - Isabelle courses included in work on thesis
  - visits at Isabelle developer team, Munich
  - support and supervision during master thesis
- **Gain experience with Formal Methods (FM)**
  - experience with Computer Algebra and FM
  - experience with Theorem Proving and FM
  - ...both at the core of future engineering
- **Produce code which will be widely useful**
  - in the Isabelle distribution worldwide
    `http://isabelle.in.tum.de/google_map.html`
  - in $\mathcal{ISAC}$, TU Graz

`https://lists.cam.ac.uk/mailman/htdig/cl-isabelle-users/2013-April/msg00020.html`

Benefits for Students

- **Get a thorough introduction to Isabelle**
  - Isabelle courses included in work on thesis
  - visits at Isabelle developer team, Munich
  - support and supervision during master thesis
- **Gain experience with Formal Methods (FM)**
  - experience with Computer Algebra and FM
  - experience with Theorem Proving and FM
  - . . . both at the core of future engineering
- **Produce code which will be widely useful**
  - in the Isabelle distribution worldwide
    `http://isabelle.in.tum.de/google_map.html`
  - in $\mathcal{ISAC}$, TU Graz

`https://lists.cam.ac.uk/mailman/htdig/cl-isabelle-users/2013-April/msg00020.html`

Benefits for Students

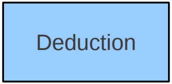
- **Get a thorough introduction to Isabelle**
  - Isabelle courses included in work on thesis
  - visits at Isabelle developer team, Munich
  - support and supervision during master thesis
- **Gain experience with Formal Methods (FM)**
  - experience with Computer Algebra and FM
  - experience with Theorem Proving and FM
  - ... both at the core of future engineering
- **Produce code which will be widely useful**
  - in the Isabelle distribution worldwide
    `http://isabelle.in.tum.de/google_map.html`
  - in $\mathcal{ISAC}$, TU Graz

`https://lists.cam.ac.uk/mailman/htdig/cl-isabelle-users/2013-April/msg00020.html`

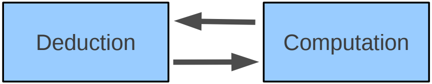
Questions?

Any questions are welcome !

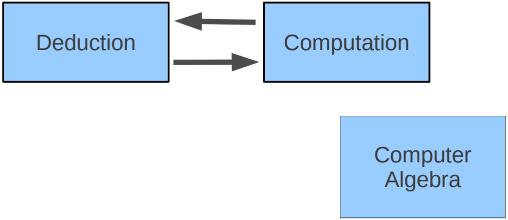
FM & CA in engineering



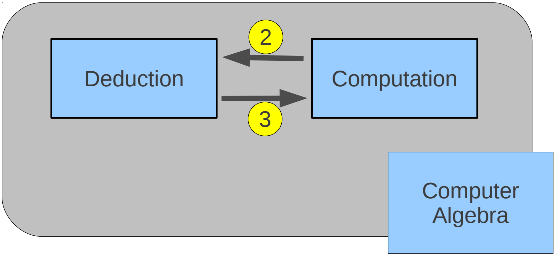
FM & CA in engineering



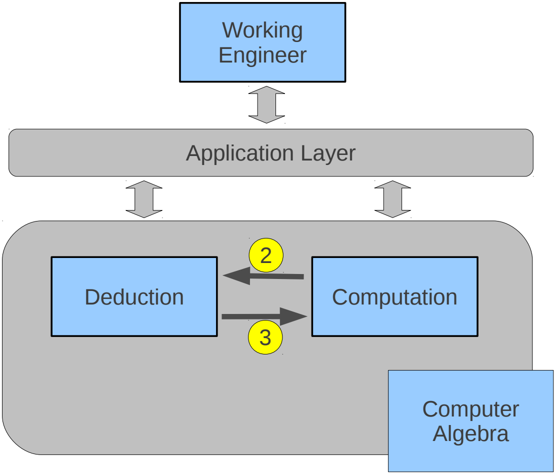
FM & CA in engineering



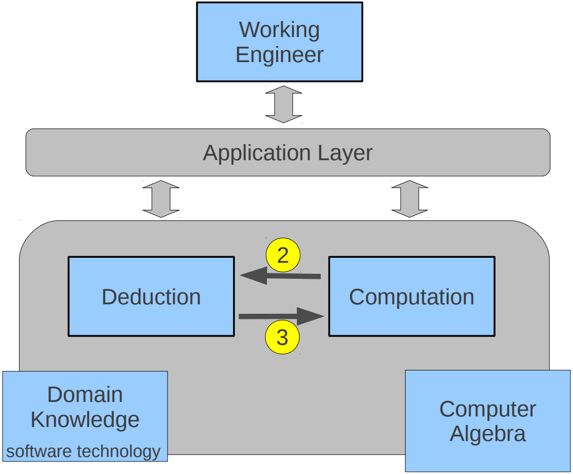
FM & CA in engineering



FM & CA in engineering



FM & CA in engineering



FM & CA in engineering

Formal Methods