DATA 61

Eisbach

A proof method language for Isabelle

Daniel Matichuk | PhD Student

October 2016

www.csiro.au

CSIRO

Isabelle Concepts

Isar, Proof Methods, and ML

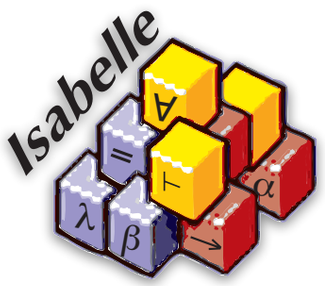
Isabelle Stack

jEdit (Scala)

Isar

Isabelle/ML

Poly/ML

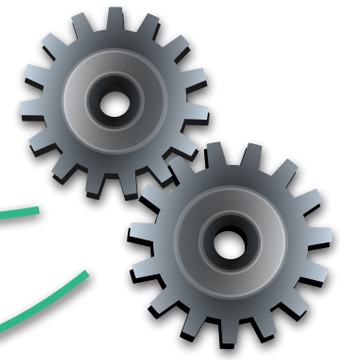


Isabelle/Isar

theorem *Knaster-Tarski'*:
  assumes *mono*[*intro*!]: $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
 shows $f\ (\bigsqcap\ \{x.\ f\ x \leq x\}) = \bigsqcap\ (\{x.\ f\ x \leq x\})$ (**is** $f\ ?a = ?a$)
proof −
 have $*$: $f\ ?a \leq ?a$ **by** (*clarsimp*,*rule order.trans*, *fastforce*)
 also have $?a \leq f\ ?a$ **by** (*fastforce intro*!: $*$)
 finally show $f\ ?a = ?a$ **.**
qed

Isabelle/Isar

theorem *Knaster-Tarski'*:
  assumes *mono*[*intro*!]: $\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y$
 shows $f\ (\bigsqcap\ \{x.\ f\ x \leq x\}) = \bigsqcap\ (\{x.\ f\ x \leq x\})$ (**is** $f\ ?a = ?a$)
proof −
 have ∗: $f\ ?a \leq ?a$ **by** (*clarsimp*,*rule order.trans*, *fastforce*)
 also have $?a \leq f\ ?a$ **by** (*fastforce intro*!: ∗)
 finally show $f\ ?a = ?a$.
qed



Proof Methods

$$\textbf{have} \ *\text{:} \ f \ ?a \leq ?a \ \textbf{by} \ (clarsimp, rule \ order.trans, \ fastforce)$$

$$\textbf{also have} \ ?a \leq f \ ?a \ \textbf{by} \ (fastforce \ intro!\text{:} \ *)$$

Proof Methods

have $*$: $f\ ?a \leq ?a$ **by** $(clarsimp, rule\ order.trans,\ fastforce)$

Goal

also have $?a \leq f\ ?a$ **by** $(fastforce\ intro!: *)$

Proof Methods

have $*$: $f\ ?a \leq ?a$ **by** $(clarsimp, rule\ order.trans,\ fastforce)$

Goal

Method

also have $?a \leq f\ ?a$ **by** $(fastforce\ intro!: *)$

Proof Methods

have $*$: $f\ ?a \leq ?a$ **by** $(clarsimp, rule\ order.trans, fastforce)$

Goal

Method

Combinator

also have $?a \leq f\ ?a$ **by** $(fastforce\ intro!: *)$

Proof Methods

Method Expression

have $*$: $f\ ?a \leq ?a$ **by** $(clarsimp, rule\ order.trans, fastforce)$

Goal

Method

Combinator

also have $?a \leq f\ ?a$ **by** $(fastforce\ intro!: *)$

Proof Methods

Proof Methods

- Syntactic layer for *tactics*

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`
- Combinators used to make *method expressions*

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`
- Combinators used to make *method expressions*
  - e.g. `apply ((subst foo | (rule baz; simp?))+)[1]`

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`
- Combinators used to make *method expressions*
  - e.g. `apply ((subst foo | (rule baz; simp?))+)[1]`
- Implemented in Isabelle/ML

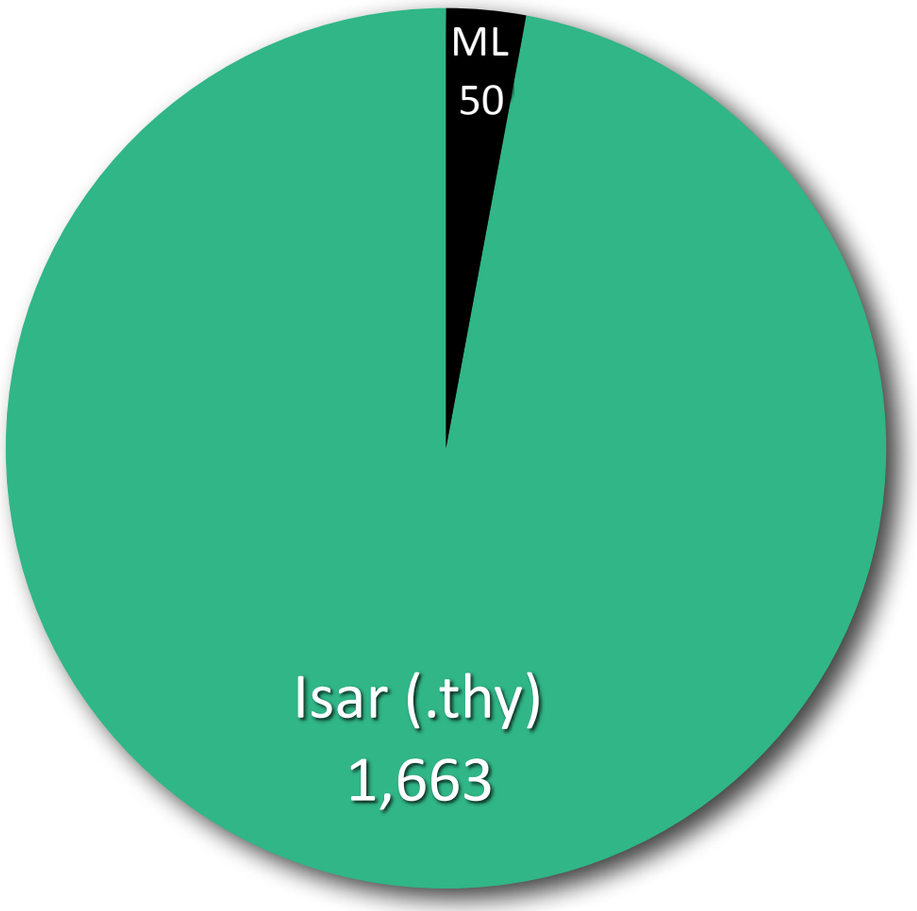
Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`
- Combinators used to make *method expressions*
  - e.g. `apply ((subst foo | (rule baz; simp?))+)[1]`
- Implemented in Isabelle/ML
  - requires knowledge of Isabelle's implementation

Proof Methods

- Syntactic layer for *tactics*
  - LCF-style reasoning to guarantees soundness
- Perform arbitrary (potentially unsafe) transformations
  - e.g. claim assumptions are contradictory
    - `apply (rule FalseE)`
- Extensible through *declaration attributes*
  - e.g. always use fact as introduction rule when applicable
    - `declare my_fact[intro!]`
- Combinators used to make *method expressions*
  - e.g. `apply ((subst foo | (rule baz; simp?))+)[1]`
- Implemented in Isabelle/ML
  - requires knowledge of Isabelle's implementation
  - often break with API changes

Isar vs. ML



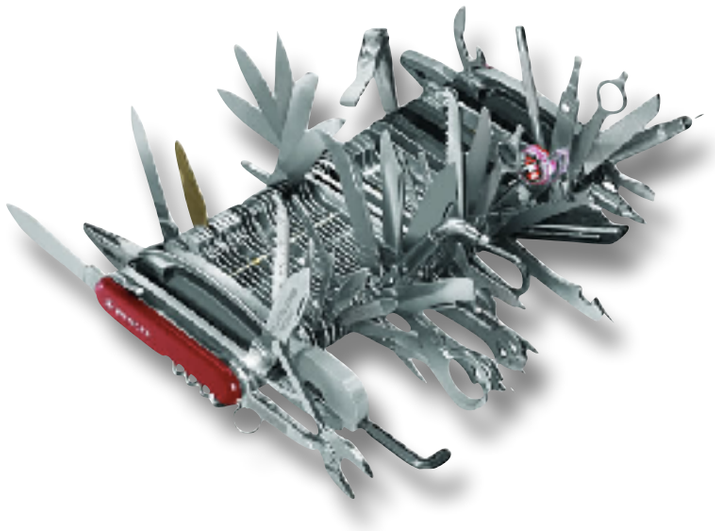
seL4

Our experience

- Full functional correctness proof
  - Open source proof and code
  - http://seL4.systems for more info

- Isabelle proof methods developed
  - wp/wpc
  - vcg for monadic hoare logic
  - sep-cancel, sep_solve …
  - automating separation logic

- Proof Engineers want more!



DATA 61

CSIRO

Eisbach

Easy Custom Proof Methods

Schwabinger Bach
Hirschau
Tennis Hirschau
Isarring
Gyßlingstraße
Eisbach
Isar
Heinrich-Mann-Allee
Pienzenauerstraße
Dehmelstraße
Mauerkircherstraße
Opitzstraße
Steubstraße
Felix-Dahn-Straße
Redwitzstraße
Flemingstraße
Reuchlinstraße
Münchner Kreis für Volksmusik Lied...
Seestadl im
HVB Club
Freizeit
BIL Leasing GmbH &
4C Catering Service

Isar


Eisbach
Isar
Schwabinger Bach
Hirschau
Tennis Hirschau
Isarring
Gyßlingstraße
Eisbach
Isar
Heinrich-Mann-Allee
Pienzenauerstraße
Opitzstraße
Dehmelstraße
Mauerkircherstraße
Steubstraße
Felix-Dahn-Straße
Flemingstraße
Reuchlinstraße
Redwitzstraße
Münchner Kreis für Volksmusik Lied...
Seestadl im ischen Garten
HVB Club
Freizeit
BIL Leasing GmbH &
4C Catering Service

Demo

Tactic languages are not new

- Ltac
  - Untyped High-level tactic language for Coq
  - Goal matching, iteration, recursion

- VeriML
  - Dependently typed tactic language
  - Provides strong static guarantees

- Mtac
  - Typed tactic language for Coq
  - Leverages built-in Coq notion of computation
  - Strong static guarantees

What distinguishes Eisbach?

- Extensive backtracking support
  - In Isabelle’s combinators and match method
- Named theorems
  - Efficient and convenient databases of facts
- Powerful pattern matching
  - Using Isabelle’s unifier
- Extensible
  - Language extensions can be built as proof methods (in ML)
  - e.g. match is simply another proof method

DATA 61

Thank You!

SSRG
Daniel Matichuk
PhD Student

e daniel.matichuk@data61.csiro.au
w ts.data61.csiro.au/people/?cn=Daniel+Matichuk

www.csiro.au

CSIRO