Interactive Simplifier Tracing and Debugging in Isabelle

Lars Hupel

Technische Universität München
Chair for Logic and Verification

July 8th, 2014

Agenda

Isabelle

- interactive proof assistant
- powerful automation
  - classical and equational reasoning
  - decision procedures (e.g. linear arithmetic)
  - integration with external automated theorem provers
  - ...
- IDE with continuous proof checking based on jEdit

Simplifier

- one of the core tactics in Isabelle
- huge: more than 1800 lines of code
- applies rewrite rules to terms
- rules can be conditional: preconditions solved recursively
- rules can be lazy: “simprocs” can generate rules on the fly
- goals can be conditional: local assumptions are used

Simplifier

- one of the core tactics in Isabelle
- huge: more than 1800 lines of code
- applies rewrite rules to terms
- rules can be conditional: preconditions solved recursively
- rules can be lazy: “simprocs” can generate rules on the fly
- goals can be conditional: local assumptions are used

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1)$$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1))$$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1))$$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1)) = 0$$

- $0 < ((x + 1) + (x + 1))$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1)) = 0$$

- $0 < ((x + 1) + (x + 1))$
  - $0 < x + 1$
  - $0 < x + 1$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1)) = 0$$

- $0 < ((x + 1) + (x + 1))$
  - $0 < x + 1$
  - $0 < x + 1$

Simplifier

Example: Conditional rewrite rules

$\mathbf{x}, \mathbf{y} \in \mathbb{N}$

$$2 \cdot x = x + x \tag{1}$$

$$x < y \implies x - y = 0 \tag{2}$$

$$0 < x + 1 \tag{3}$$

$$0 < x \implies 0 < y \implies 0 < x + y \tag{4}$$

$$0 - 2 \cdot (x + 1) = 0 - ((x + 1) + (x + 1)) = 0$$

- $0 < ((x + 1) + (x + 1))$
  - $0 < x + 1$
  - $0 < x + 1$

Simplifier

Simplification might go wrong:

- no result at all
- unexpected result
- non-termination

Simplifier

Simplification might go wrong:

- no result at all
- unexpected result
- non-termination

tackled by tracing

Simplifier Trace

Lists all rewriting steps, but:

- potentially huge
- can't be filtered (e.g. "trace only applications of X and Y")
- offers no hierachical structure
- problematic with non-termination

```
[1]SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
10 ^ 5 * 10 ^ 5 = 10 ^ 10
[1]Applying instance of rewrite rule "Power.semiring_numeral_class.power_numeral"
numeral ?k1 ^ numeral ?l1 ≡ numeral (Num.pow ?k1 ?l1)
[1]Rewriting:
10 ^ 5 ≡ numeral (Num.pow (num.Bit0 (num.Bit1 (num.Bit0 num.One))) (num.Bit1 (nu
[1]Applying instance of rewrite rule "Num.pow.simps_3":
Num.pow ?x1 (num.Bit1 ?y1) ≡ Num.sqr (Num.pow ?x1 ?y1) * ?x1
[1]Rewriting:
Num.pow (num.Bit0 (num.Bit1 (num.Bit0 num.One))) (num.Bit1 (num.Bit0 num.One)) ≡
Num.sqr (Num.pow (num.Bit0 (num.Bit1 (num.Bit0 num.One))) (num.Bit0 num.One)) * r
[1]Applying instance of rewrite rule "Num.pow.simps_2":
Num.pow ?x1 (num.Bit0 ?y1) ≡ Num.sqr (Num.pow ?x1 ?y1)
[1]Rewriting:
Num.pow (num.Bit0 (num.Bit1 (num.Bit0 num.One))) (num.Bit0 num.One) ≡ Num.sqr (N
[1]Applying instance of rewrite rule "Num.pow.simps_1":
Num.pow ?y num.One ≡ ?y
[1]Rewriting:
Num.pow (num.Bit0 (num.Bit1 (num.Bit0 num.One))) num.One ≡ num.Bit0 (num.Bit1 (n
```

Agenda

Overview

- interactive
- breakpoints on terms and theorems
- configurable verbosity
- integrated into Isabelle/jEdit

Demonstration

Related Work

SWI-Prolog

- offers interactive tracing
- supports breakpoints
- speciality: marking goals as success

Related Work

SWI-Prolog

- offers interactive tracing
- supports breakpoints
- speciality: marking goals as success
  - In Isabelle: difficult because of proof kernel

Related Work

Maude

- offers interactive tracing
- supports breakpoints
- speciality: during rewriting, issue new goal

Related Work

Maude

- offers interactive tracing
- supports breakpoints
- speciality: during rewriting, issue new goal
  - In Isabelle: rarely needed because of parallel processing

Agenda

Challenges

Selective Memory Clearing

Scenario

1. rewrite step fails
2. user chooses to redo the step
3. simplification starts anew
4. memoization kicks in, step fails again

Challenges

Selective Memory Clearing

Scenario

1. rewrite step fails
2. user chooses to redo the step
3. simplification starts anew
4. memoization kicks in, step fails again

Challenges

Context Handling

- simplification result depends on local assumptions
- memoization might not make sense across different contexts

$$(P \Longrightarrow P) \Longrightarrow (Q \Longrightarrow P) \Longrightarrow R$$

Challenges

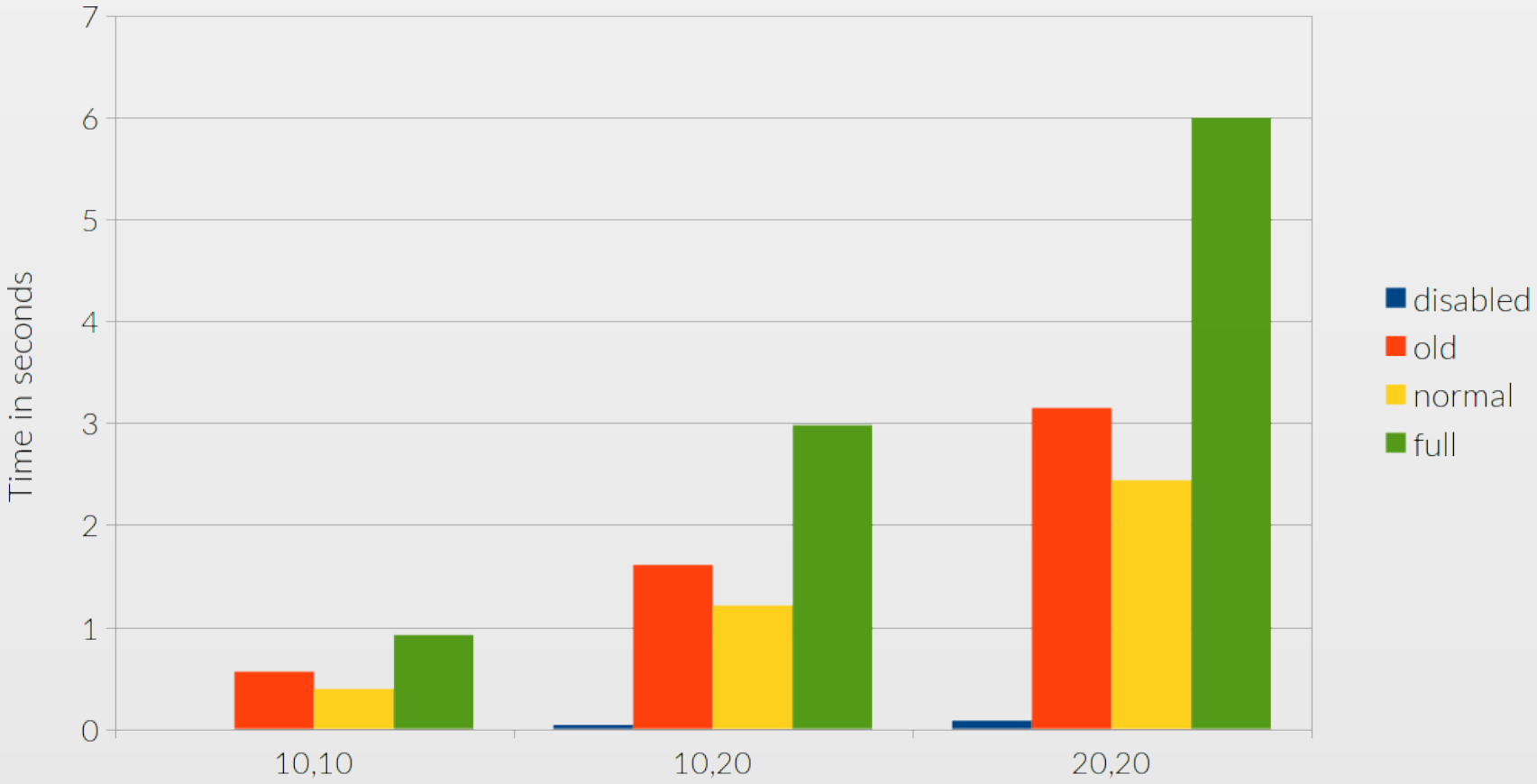
User Experience

- user feedback is generally positive
- used for detecting erratic rules, analyzing simplifier runtime, …
- very flexible, but: every additional option generates confusion

Agenda

Performance

Simplifying $10^x \cdot 10^y$



Test machine: Core i7, 3.7 GHz

A Parallelized Simplifier?

- tracing is completely asynchronous
- supports multiple questions at the same time
- but: unused by the simplifier
- proof of concept: development of a tiny, parallel simplifier

A Parallelized Simplifier?

Lessons Learned

Advantages

- almost trivial to implement for a toy simplifier
- GUI part works out of the box

A Parallelized Simplifier?

Lessons Learned

Advantages

- almost trivial to implement for a toy simplifier
- GUI part works out of the box

Disadvantages

- potentially confusing for users
  - lots of spurious messages
  - better filtering required?
  - holding back messages required?

Outcomes

- a *generic* tracing facility
  - using its interface requires little changes to a tactic
  - parallelization-ready
  - but not 100% there yet
- first steps towards instrumenting the simplifier
  - Should all tactics be written in continuation-passing style?

Future Work

- support for more tactics
- support for other traces (unifier, simp debug, …)
- memoization: fuzzy matching
- term provenance (“Where does that ‘5’ come from?”)
- tighter integration into Isabelle/jEdit

Q & A