Generalising the One-Point Rule

Johannes Hölzl

VU Amsterdam

Matryoshka Workshop 2018

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied

Lean’s Simplifier

Lean’s simplifier is similar to Isabelle’s

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++
- **no `simp`-procs** (rewrite tactics used when a pattern is detected)
  Used in Isabelle for:

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++
- **no `simp`-procs** (rewrite tactics used when a pattern is detected)
  Used in Isabelle for:
  - cancellation: $a + b + \mathbf{c} + d = e + \mathbf{c} \leftrightarrow a + b + d = e$

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++
- **no `simp`-procs** (rewrite tactics used when a pattern is detected)
  Used in Isabelle for:
  - cancellation: $a + b + \mathbf{c} + d = e + \mathbf{c} \leftrightarrow a + b + d = e$
  - unit replacement: $u = ()$

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++
- **no `simp`-procs** (rewrite tactics used when a pattern is detected)
  Used in Isabelle for:
  - cancellation: $a + b + \mathbf{c} + d = e + \mathbf{c} \leftrightarrow a + b + d = e$
  - unit replacement: $u = ()$
  - lin. arith.: $2 < x \rightarrow (3 * x + 1 < 4 * x) \leftrightarrow \texttt{true}$

Lean's Simplifier

Lean's simplifier is similar to Isabelle's

- Rewriting until no further rw-rule can be applied
- Conditional rewrite rules
- Detect permutation rule (Perm rules only apply when term gets smaller)
- Implemented in C++
- **no `simp`-procs** (rewrite tactics used when a pattern is detected)
  Used in Isabelle for:
  - cancellation: $a + b + \mathbf{c} + d = e + \mathbf{c} \leftrightarrow a + b + d = e$
  - unit replacement: $u = ()$
  - lin. arith.: $2 < x \to (3 * x + 1 < 4 * x) \leftrightarrow \mathtt{true}$
  - one-point rule $(\exists x, p\ x \wedge x = t) \leftrightarrow p\ t$

Notes about the simplifier

- Easy to extend (add your own rules, `simp`-procs)

Notes about the simplifier

- Easy to extend (add your own rules, `simp`-procs)
- Provide theorem collection (e.g. Isabelle's `field-simps`)

Notes about the simplifier

- Easy to extend (add your own rules, `simp`-procs)
- Provide theorem collection (e.g. Isabelle's `field-simps`)
- Powerful by combining very simple rules, methods, ...

Notes about the simplifier

- Easy to extend (add your own rules, `simp`-procs)
- Provide theorem collection (e.g. Isabelle's `field-simps`)
- Powerful by combining very simple rules, methods, ...
- Not necessary to solve goal
  User often sees how to continue

WIP: Simp loop in Lean

Loop obvious implementation:

- Lean’s simplifier

WIP: Simp loop in Lean

Loop obvious implementation:

- Lean's simplifier
- Lean's definitional simplifier

WIP: Simp loop in Lean

Loop obvious implementation:

- Lean's simplifier
- Lean's definitional simplifier
- Try to apply a list of simp procs

WIP: Simp loop in Lean

Loop obvious implementation:

- Lean’s simplifier
- Lean’s definitional simplifier
- Try to apply a list of simp procs
- Terminate when nothing changed

The One-Point Rule

$(\exists x, p\ x \land x = t) \leftrightarrow p\ t$, or

$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$

The One-Point Rule

$(\exists x, p\ x \wedge x = t) \leftrightarrow p\ t$, or

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N})(z : \mathbb{Z}), p\ z \wedge \exists (x : \texttt{vec}\ \alpha\ n)(y : \mathbb{Z}), q\ x\ y \wedge n = t\ z\ y$$

The One-Point Rule

$$(\exists x, p\ x \land x = t) \leftrightarrow p\ t, \text{ or}$$

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N})(z : \mathbb{Z}), p\ z \land \exists (x : \texttt{vec}\ \alpha\ n)(y : \mathbb{Z}), q\ x\ y \land n = t\ z\ y$$

- Provides basic quantifier elimination

The One-Point Rule

$$(\exists x, p\ x \land x = t) \leftrightarrow p\ t, \text{ or}$$

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N}) (z : \mathbb{Z}), p\ z \land \exists (x : \texttt{vec}\ \alpha\ n) (y : \mathbb{Z}), q\ x\ y \land n = t\ z\ y$$

- Provides basic quantifier elimination
- Important to handle logic, e.g.

The One-Point Rule

$$(\exists x, p\ x \wedge x = t) \leftrightarrow p\ t, \text{ or}$$

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N})(z : \mathbb{Z}), p\ z \wedge \exists (x : \texttt{vec}\ \alpha\ n)(y : \mathbb{Z}), q\ x\ y \wedge n = t\ z\ y$$

- Provides basic quantifier elimination
- Important to handle logic, e.g.
  - Image of a function:

$$\begin{array}{lcc} \forall y \in f[X], p\ y & \leftrightarrow & \text{mem. image} \\ \forall y, (\exists x \in X, y = f x) \to p\ y) & \leftrightarrow & \text{exists elim} \\ \forall y, \forall x \in X, y = f x \to p\ y & \leftrightarrow & \text{one-point rule} \\ (\forall x \in X, p\ (f x)) & & \end{array}$$

The One-Point Rule

$$(\exists x, p\ x \land x = t) \leftrightarrow p\ t, \text{ or}$$

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N})(z : \mathbb{Z}), p\ z \land \exists (x : \texttt{vec}\ \alpha\ n)(y : \mathbb{Z}), q\ x\ y \land n = t\ z\ y$$

- Provides basic quantifier elimination
- Important to handle logic, e.g.
  - Image of a function:

$$\begin{array}{lcc} \forall y \in f[X], p\ y & \leftrightarrow & \text{mem. image} \\ \forall y, (\exists x \in X, y = f x) \to p\ y) & \leftrightarrow & \text{exists elim} \\ \forall y, \forall x \in X, y = f x \to p\ y & \leftrightarrow & \text{one-point rule} \\ (\forall x \in X, p\ (f x)) & & \end{array}$$

  - Encodings of inductive predicates

The One-Point Rule

$$(\exists x, p\ x \wedge x = t) \leftrightarrow p\ t,\ \text{or}$$

$$(\forall x, x = t \to p\ x) \leftrightarrow p\ t$$

- Example:

$$\exists (n : \mathbb{N})(z : \mathbb{Z}), p\ z \wedge \exists (x : \texttt{vec}\ \alpha\ n)(y : \mathbb{Z}), q\ x\ y \wedge n = t\ z\ y$$

- Provides basic quantifier elimination
- Important to handle logic, e.g.
  - Image of a function:

$$\begin{array}{lcc} \forall y \in f[X], p\ y & \leftrightarrow & \text{mem. image} \\ \forall y, (\exists x \in X, y = f x) \to p\ y) & \leftrightarrow & \text{exists elim} \\ \forall y, \forall x \in X, y = f x \to p\ y & \leftrightarrow & \text{one-point rule} \\ (\forall x \in X, p\ (f x)) & & \end{array}$$

  - Encodings of inductive predicates
- Hope: $p\ t$ can be further simplified

Problem: Filters

Filters represent topological limits.

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

$$(\exists \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s) \to p\ s$$

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

$$(\exists \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s) \to p\ s$$

$$\forall \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s \to p\ s$$

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

$$(\exists \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s) \to p\ s$$

$$\forall \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s \to p\ s$$

Obviously, can we reduce this to $\forall \varepsilon > 0, p\ \mathcal{B}(x, \varepsilon)$?

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

$$(\exists \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s) \to p\ s$$

$$\forall \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s \to p\ s$$

Obviously, can we reduce this to $\forall \varepsilon > 0, p\ \mathcal{B}(x, \varepsilon)$?

We need to proof monotonicity of p.

Problem: Filters

Filters represent topological limits.

$$s \in \mathtt{nhds}(x) \to p\ s$$

$$(\exists \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s) \to p\ s$$

$$\forall \varepsilon > 0, \mathcal{B}(x, \varepsilon) \subseteq s \to p\ s$$

Obviously, can we reduce this to $\forall \varepsilon > 0, p\ \mathcal{B}(x, \varepsilon)$?

We need to proof monotonicity of p.

In most cases monotonicity can be proved syntactically.

WIP: Generalising to an Order Relation

$$\frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\exists x, x \leq t \wedge p\ x) \leftrightarrow p\ t} \qquad \frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\forall x, x \geq t \rightarrow p\ x) \leftrightarrow p\ t}$$

- Only requirement on $\leq$: reflexivity

WIP: Generalising to an Order Relation

$$\frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\exists x, x \leq t \land p\ x) \leftrightarrow p\ t} \qquad \frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\forall x, x \geq t \rightarrow p\ x) \leftrightarrow p\ t}$$

- Only requirement on $\leq$: reflexivity
- Application: sets, filters, simple arithemtic problems, ...

WIP: Generalising to an Order Relation

$$\frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\exists x, x \leq t \land p\ x) \leftrightarrow p\ t} \qquad \frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\forall x, x \geq t \rightarrow p\ x) \leftrightarrow p\ t}$$

- Only requirement on $\leq$: reflexivity
- Application: sets, filters, simple arithemtic problems, ...
- $\leq$ could also be a relation with two different types, but then

WIP: Generalising to an Order Relation

$$\frac{\texttt{monotone}_{\leq,\to}\ p}{(\exists x, x \leq t \wedge p\ x) \leftrightarrow p\ t} \qquad \frac{\texttt{monotone}_{\leq,\to}\ p}{(\forall x, x \geq t \to p\ x) \leftrightarrow p\ t}$$

- Only requirement on $\leq$: reflexivity
- Application: sets, filters, simple arithemtic problems, ...
- $\leq$ could also be a relation with two different types, but then
  - Find a r and a p' where $x \leq t \to r\ (p\ x)\ (p'\ t)$ and an x', s.t. $x' \leq t$

WIP: Generalising to an Order Relation

$$\frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\exists x, x \leq t \land p\ x) \leftrightarrow p\ t} \qquad \frac{\texttt{monotone}_{\leq,\rightarrow}\ p}{(\forall x, x \geq t \rightarrow p\ x) \leftrightarrow p\ t}$$

- Only requirement on $\leq$: reflexivity
- Application: sets, filters, simple arithemtic problems, ...
- $\leq$ could also be a relation with two different types, but then
  - Find a r and a p' where $x \leq t \rightarrow r\ (p\ x)\ (p'\ t)$ and an x', s.t. $x' \leq t$
  - Then we have $\texttt{related}\ (\leq)\ (\rightarrow)\ p\ p' \rightarrow (\exists x, x \leq t \land p\ x) \leftrightarrow p'\ t$

Proving Monotonicity

$$\texttt{monotone}_{\leq_1,\leq_2}\ p :\leftrightarrow (\forall x y, x \leq_1 y \to p\ x \leq_2 p\ y)$$

Proving Monotonicity

$$\texttt{monotone}_{\leq_1,\leq_2}\ p :\leftrightarrow (\forall x y, x \leq_1 y \to p\ x \leq_2 p\ y)$$

- The user provides a rule for each function c:

$$\begin{array}{llll} \forall g\ (\leq_1), & \texttt{monotone}_{\leq_1,\leq_2}\ g & \to & \texttt{monotone}_{\leq_1,\leq_3}\ (\lambda x, c\ (g\ x)) \\ \forall g\ h\ (\leq_1), & \texttt{monotone}_{\leq_1,\leq_2}\ g \to \texttt{monotone}_{\leq_1,\leq_3}\ h & \to & \texttt{monotone}_{\leq_1,\leq_4}\ (\lambda x, g\ x + h\ x) \end{array}$$

Proving Monotonicity

$$\texttt{monotone}_{\leq_1,\leq_2}\ p :\leftrightarrow (\forall xy, x \leq_1 y \to p\ x \leq_2 p\ y)$$

- The user provides a rule for each function c:

$$\begin{array}{llll} \forall g\ (\leq_1), & \texttt{monotone}_{\leq_1,\leq_2}\ g & \to & \texttt{monotone}_{\leq_1,\leq_3}\ (\lambda x, c\ (g\ x)) \\ \forall g\ h\ (\leq_1), & \texttt{monotone}_{\leq_1,\leq_2}\ g \to \texttt{monotone}_{\leq_1,\leq_3}\ h & \to & \texttt{monotone}_{\leq_1,\leq_4}\ (\lambda x, g\ x + h\ x) \end{array}$$

- Do backwards search. HO-unification instantiates g (and h...) with identity at the leafs

Focusing (Balancing)

Often x appears in a term, so we focus on it:

Focusing (Balancing)

Often x appears in a term, so we focus on it:

Galois connections: $f\ x \ \leq_1\ y \quad \leftrightarrow \quad x\ \leq_2\ g\ y$

Focusing (Balancing)

Often x appears in a term, so we focus on it:

Galois connections:	$f\ x \leq_1 y \leftrightarrow x \leq_2 g\ y$
„Injectivity”:	$f\ x \leq_1 g\ y \leftrightarrow x \leq_2 y$

Focusing (Balancing)

Often x appears in a term, so we focus on it:

Galois connections:	$f\ x \leq_1 y \leftrightarrow x \leq_2 g\ y$
„Injectivity”:	$f\ x \leq_1 g\ y \leftrightarrow x \leq_2 y$
Splitting rules:	$f\ x \leq_1 t \leftrightarrow (C_1 \wedge x \leq_2 t_1) \vee (C_2 \wedge x \leq_3 t_2)$

Collecting for Semilattices

- Collect upper (or lower) bounds

Collecting for Semilattices

- Collect upper (or lower) bounds
- e.g. $(x \leq t \wedge x \leq s) \leftrightarrow x \leq t \sqcap s$

Collecting for Semilattices

- Collect upper (or lower) bounds
- e.g. $(x \leq t \wedge x \leq s) \leftrightarrow x \leq t \sqcap s$
- is important, only $t \leq x$ is monotone in x, $x \leq t$ isn't

Final Slide

- Generalized one-point rule
- No implementation / evaluation yet
- Ideas for other (easy) `simp`-procs

Final Slide

- Generalized one-point rule
- No implementation / evaluation yet
- Ideas for other (easy) `simp`-procs

Thanks for listening! Questions?