Proof-Relevant Rewriting Strategies (WIP)
Matthieu Sozeau – Inria Paris & PPS

6th Coq Workshop
July 17th 2014
Vienna, Austria
Generalized Rewriting

- **Equational reasoning** $x = y \vdash x + 1 \Rightarrow y + 1$
- **Logical reasoning** $x \leftrightarrow y \vdash (x \land y) \Rightarrow (x \land x)$
- **Rewriting** $y \Rightarrow z \vdash x \Rightarrow y \Rightarrow x \Rightarrow z$
- **Abstract data types, quotients/setoids**
  - $s, t : list, x =set y \vdash \text{union } x y =set x \Rightarrow \text{union } x x =set x$
Rewriting in Type Theory

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality.
  \[ \Pi A (P : A \rightarrow \text{Type}) \ (x \ y : A), \ P \ x \rightarrow x = y \rightarrow P \ y. \]
  - ✔ Applies to any context
  - ✗ Large proof term: repeats the context that depends on \( x \)
  - ✗ Restricted to equality, one rewrite at a time

One can build a set of combinators to rewrite in depth: HOL conversions [Paulson 83].
Moving from substitution to congruence.

- **Built-in substitution: Leibniz equality.**
  \[ \Pi A \ (P : A \rightarrow \text{Type}) \ (x \ y : A), \ P \ x \rightarrow x = y \rightarrow P \ y. \]
  - ✔ Applies to any context
  - ✗ Large proof term: repeats the context that depends on \( x \)
  - ✗ Restricted to equality, one rewrite at a time

- **Congruence.**
  \[ \Pi A \ B \ (f : A \rightarrow B) \ (x \ y : A), \ x = y \rightarrow f \ x = f \ y \]
  - ✗ Applies at the toplevel only
  - ✔ Small proof term: mentions the changed terms only
  - ✔ Generalizes to n-ary, parallel rewriting
  - ✗ Still restricted to equality
Moving from substitution to congruence.

- Built-in substitution: Leibniz equality.
  \[ \Pi A (P : A \to \text{Type}) (x y : A), P x \to x = y \to P y. \]
  - ✓ Applies to any context
  - ✗ Large proof term: repeats the context that depends on \( x \)
  - ✗ Restricted to equality, one rewrite at a time

- Congruence.
  \[ \Pi A B (f : A \to B) (x y : A), x = y \to f x = f y \]
  - ✗ Applies at the toplevel only
  - ✓ Small proof term: mentions the changed terms only
  - ✓ Generalizes to \( n \)-ary, parallel rewriting
  - ✗ Still restricted to equality

One can build a set of combinators to rewrite in depth: HOL conversions [Paulson 83].
Generalized Rewriting in Type Theory

Basin [NuPRL, 94], Sacerdoti Coen [CoQ, 04]

- Generalized to any relation
  \( \text{Proper (iff } \leftrightarrow \text{ iff) not } \triangleq \prod P Q, P \leftrightarrow Q \rightarrow \neg P \leftrightarrow \neg Q \)

- Multiple signatures for a given constant
  \( \text{Proper (impl } \rightarrow \rightarrow \text{ impl) not} \)
Generalized Rewriting in Type Theory

Basin [NuPRL, 94], Sacerdoti Coen [CoQ, 04]

- Generalized to any relation
  \[ \text{Proper } (\text{iff } \leftrightarrow \text{iff}) \not\equiv \Pi \ P, \ P \leftrightarrow Q \rightarrow \neg P \leftrightarrow \neg Q \]

- Multiple signatures for a given constant
  \[ \text{Proper } (\text{impl } \rightarrow \rightarrow \text{impl}) \not\equiv \]

Requires proof search:

- Heuristic in NuPRL based on subrelations (impl \subset \iff)
- Complete procedure in CoQ.

Both are monolithic algorithms with a primitive notion of signature: a list of atomic relations (with variance).
Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

\[
\begin{align*}
\text{all} & : \forall A : \text{Type}, (A \to \text{Prop}) \to \text{Prop} \\
\Pi A, \text{Proper} \ (\text{pointwise	extunderscore relation} \ A \text{ iff } \leftrightarrow \text{ iff}) (@\text{all} \ A)
\end{align*}
\]
Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)
  \[ \forall A : \text{Type}, (A \to \text{Prop}) \to \text{Prop} \]
  \[ \Pi A, \text{Proper} (\text{pointwise relation } A \iff \leftrightarrow \iff) (@\text{all } A) \]

- An algebraic presentation, supporting higher-order functions (rewriting under binders) and polymorphism:
  \[ \Pi A B C R_0 R_1 R_2, \]
  \[ \text{Proper} ((R_1 \leftrightarrow R_2) \leftrightarrow (R_0 \leftrightarrow R_1) \leftrightarrow (R_0 \leftrightarrow R_2)) \]
  \[ (@\text{compose } A B C) \]
A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)
  \[
  \text{all} : \forall A : \text{Type}, (A \rightarrow \text{Prop}) \rightarrow \text{Prop}
  \]
  \[
  \Pi A, \text{Proper} (\text{pointwise\_relation } A \iff \leftrightarrow \iff) (@\text{all } A)
  \]

- An algebraic presentation, supporting higher-order functions (rewriting under binders) and polymorphism:
  \[
  \Pi A B C R_0 R_1 R_2, \quad \text{Proper} ((R_1 \leftrightarrow R_2) \leftrightarrow (R_0 \leftrightarrow R_1) \leftrightarrow (R_0 \leftrightarrow R_2))
  \]
  \[
  (@\text{compose } A B C)
  \]

- Generic morphism declarations.
A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)
  \[ \forall A : \text{Type}, \ (A \to \text{Prop}) \to \text{Prop} \]
  \[ \Pi A, \ \text{Proper} \ (\text{pointwise relation} \ A \iff \leftrightarrow \iff) \ (@\text{all} \ A) \]

- An algebraic presentation, supporting higher-order functions (rewriting under binders) and polymorphism:
  \[ \Pi A B C R_0 R_1 R_2, \]
  \[ \ \text{Proper} \ ((R_1 \leftrightarrow R_2) \leftrightarrow (R_0 \leftrightarrow R_1) \leftrightarrow (R_0 \leftrightarrow R_2)) \]
  \[ (@\text{compose} \ A B C) \]

- Generic morphism declarations.

- Support for subrelations, quotienting the signatures.
Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)
  \[ \forall A : \text{Type}, \ (A \rightarrow \text{Prop}) \rightarrow \text{Prop} \]
  \[ \Pi \ A, \ \text{Proper} \ (\text{pointwise relation} \ A \ \text{iff} \ +\rightarrow \ \text{iff}) \ (@\text{all} \ A) \]

- An algebraic presentation, supporting higher-order functions (rewriting under binders) and polymorphism:
  \[ \Pi \ A \ B \ C \ R_0 \ R_1 \ R_2, \]
  \[ \text{Proper} \ ((R_1 \ +\rightarrow \ R_2) \ +\rightarrow \ (R_0 \ +\rightarrow \ R_1) \ +\rightarrow \ (R_0 \ +\rightarrow \ R_2)) \]
  \[ (@\text{compose} \ A \ B \ C) \]

- Generic morphism declarations.

- Support for subrelations, quotienting the signatures.

- Rewriting on operators/functions, parallel rewrites...
Class Proper \{A\} (R : relation A) (m : A) : Prop := proper : R m m.

Instance reflexive_proper ‘(Reflexive A R) (x : A) : Proper R x.
Class Proper \(\{A\} (R : \text{relation } A) (m : A) : \text{Prop} \) :=
proper : \(R \; m \; m\).

Instance reflexive_proper \((\text{Reflexive } A \; R) (x : A) : \text{Proper } R \; x\).

Definition respectful \(\{A \; B : \text{Type}\} \)
\((R : \text{relation } A) (R' : \text{relation } B) : \text{relation } (A \rightarrow B) :=
\text{fun } f \; g \Rightarrow \forall \; x \; y, \; R \; x \; y \rightarrow R' \; (f \; x) \; (g \; y)\).
Class Proper \{A\} (R : relation A) (m : A) : Prop := 
proper : R m m.

Instance reflexive_proper ‘(Reflexive A R) (x : A) : Proper R x.

Definition respectful \{A B : Type\}
(R : relation A) (R' : relation B) : relation (A \to B) := 
fun f g \Rightarrow \forall x y, R x y \to R' (f x) (g y).

Notation " R ++> R' " := (respectful R R') (right associativity).
Notation " R --> R' " := (R^{-1} ++> R') (right associativity).
Class Proper \{A\} (R : relation A) (m : A) : Prop := proper : R m m.

Instance reflexive_proper ‘(Reflexive A R) (x : A) : Proper R x.

Definition respectful \{A B : Type\}
  (R : relation A) (R' : relation B) : relation (A → B) :=
  fun f g ⇒ ∀ x y, R x y → R' (f x) (g y).

Notation " " R ++> R’ " := (respectful R R') (right associativity).

Notation " " R →→ R’ " := (R⁻¹ ++> R’) (right associativity).

Instance not_P : Proper (iff ++> iff) not.
1. Generalized Rewriting in Type Theory
2. Proof-relevant relations
3. Rewriting Strategies
Impredicativity helped

All fine with relations in \textit{Prop}, how about \textit{Type}-valued relations?

\[
\text{Proper} : \Pi A : \text{Type}_i, (A \to A \to \text{Type}_j) \to A \to \text{Type}_j.
\]

Need to show, under \(A : \text{Type}_i:\)

\[
\text{Proper} \quad ((A \to A \to \text{Type}_j) \to A \to \text{Type}_j) \\
(\text{iso}_\text{rel} \ A \to \text{eq} \ A \to \text{iso}) \\
(\text{Proper} \ A)
\]

Inconsistency: \(\text{Type}_{\max(i,j+1)} \not\subseteq \text{Type}_i\)
With full universe polymorphism (Sozeau & Tabareau [ITP'14]):

$$\text{Proper}_{ij} : \Pi A : \text{Type}_i, (A \rightarrow A \rightarrow \text{Type}_j) \rightarrow A \rightarrow \text{Type}_j$$

We can show, under $A : \text{Type}_i$:

$$\text{Proper}_{i',j'} \ ((A \rightarrow A \rightarrow \text{Type}_j) \rightarrow A \rightarrow \text{Type}_j) \ (\text{iso}_\text{rel} A \rightarrow \text{eq} A \rightarrow \text{iso}) \ (\text{Proper}_{ij} A)$$

With constraint: $\max(i, j + 1) \leq i'$. 

Actually, $\text{crelation}(A : \text{Type}_i) := A \rightarrow A \rightarrow \text{Type}_j$ is already problematic: no relation equivalence or subrelation definition possible.
Generalized rewriting now handles:

▶ The function space “relation”: rewrite $x$ to $y$ in $C$ builds
  $prf : C[x] \rightarrow C[y]$

▶ Isomorphism of types

▶ Computationally relevant relations, e.g. CoRN’s appartness
  relation on reals.

▶ Hom-types of categories which are not $Prop$-based setoids,
  e.g. groupoids.
1 Generalized Rewriting in Type Theory

2 Proof-relevant relations

3 Rewriting Strategies
Automated rewriting

An efficiency concern: autorewrite does repeat rewrite.

- Crawls through the whole goal each time.
- Applies transitivity of rewriting at the top-level only, resulting in large proof-terms.

We want to allow the specification of precise rewriting strategies (e.g. bottomup, innermost, repeated...) that avoid this.

- Traversal of the goal specified by the user.
- Applies transitivity of rewriting at inner points of the term, resulting in shorter proof-terms.
Based on ELAN’s rewriting strategies

Implemented using the LogicT monad (failure/success continuations) for efficient backtracking and clear semantics.

Using the existing generalized rewriting framework to produce Proper constraints and build the rewriting proofs.

Interface: rewrite_strat strategy (in t)?
Rewriting strategies

\( s, t, u ::= (\leq)? c \) (right to left?) lemma

- fail | id failure | identity
- refl reflexivity
- progress \( s \) progress
- try \( s \) failure catch
- \( s ; u \) composition
- \( s \lvert\lvert t \) left-biased choice
- repeat \( s \) iteration (+)
- subterm(s)? \( s \) one or all subterms
- innermost \( s \) innermost first
- hints \( hintdb \) apply first matching hint
- eval \( redexpr \) apply reduction
- fold \( c \) fold expression
- pattern \( p \) pattern matching
try $s$ = $s || \text{id}$

any $s$ = fix $u$.try $(s ; u)$

repeat $s$ = $s ; \text{any } s$

bottomup $s$ = fix $bu$.((progress (subterms $bu$)) || $s$) ; try $bu$

topdown $s$ = fix $td$.($s \parallel (\text{progress (subterms } td))) ) ; \text{try } td$

innermost $s$ = fix $i$.((subterm $i$) || $s$)

outermost $s$ = fix $o$.($s \parallel (\text{subterm } o)$)
Suppose the theory of monoids on $T$.
A goal: $x y : T \vdash x \bullet (\epsilon \bullet y) \bullet \epsilon$.

- autorewrite with monoids will do two rewrites with both unit laws, the proof term will be roughly twice the goal size.
- rewrite_strat (topdown (repeat (hints monoids))) will first rewrite $\epsilon \bullet y$ to $y$ and directly after, $y \bullet \epsilon$ to $y$, resulting in a proof term of size roughly that of the initial goal, and will be twice as fast as well.
Advantages

- Improved performance by replacing autorewrite tactic used in Ring with: topdown (hints *Esimpl*)

- Avoid mixing of rewrite with Ltac constructs, e.g.:
  
  \[(\text{rewrite } l_1 \mid\ldots\mid\text{progress rewrite } l_n)\]
  
  becomes

  \[\text{rewrite\_strat} (l_1 \mid\ldots\mid\text{progress } l_n)\]
  
  which traverses the term just once.

- Another common pattern:

  match goal with
  
  \[\vdash\text{context } [t] \Rightarrow\text{rewrite } l\]
  
  end

  =

  rewrite\_strat (topdown (pattern t; term l))
Future work

- Debug & release
- Benchmarks
The End