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

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

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

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

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

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

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

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

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)

\[
\forall A : \text{Type}, (A \to \text{Prop}) \to \text{Prop}
\]

\[
\Pi A, \text{Proper} (\text{pointwise\_relation } A \iff \leftrightarrow \text{iff}) (@\text{all } A)
\]
A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)
  
  \( \forall A : \text{Type}, (A \to \text{Prop}) \to \text{Prop} \)
  
  \( \Pi A, \text{Proper} (\text{pointwise\_relation}\ A \iff \quad +\to \quad \iff) (\@\text{all}\ A) \)

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

Sozeau [JFR 2009]

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

- An algebraic presentation, supporting higher-order functions (rewriting under binders) and polymorphism:

  \[
  \Pi A B C R_0 R_1 R_2,
  \]
  
  \[
  \text{Proper} \ ((R_1 + + R_2) + + (R_0 + + R_1) + + (R_0 + + R_2))
  \]
  
  \[
  (@\text{compose } A B C)
  \]

- Generic morphism declarations.
A New Look at Generalized Rewriting

Sozeau [JFR 2009]

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

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

- Generic morphism declarations.

- Support for subrelations, quotienting the signatures.
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.

- 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 : 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).
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 \"\(\rightarrow\) \(\rightarrow\) R' \" := (respectful R R') (right associativity).
Notation \"\(\rightarrow\) R' \" := (R⁻¹ \(\rightarrow\) 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) :=
\[
\text{fun } f g \Rightarrow \forall x y, R x y \rightarrow R' (f x) (g y).
\]

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 Prop, how about 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 $A : \text{Type}_i$:

$$\text{Proper} \quad ((A \to A \to \text{Type}_j) \to A \to \text{Type}_j)$$

$$\quad (\text{iso} \_ \text{rel} \ A \to \text{eq} \ A \to \text{iso})$$

$$\quad (\text{Proper} \ A)$$

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

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

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

\[ \text{Proper}_{i' \, j'} \quad ((A \to A \to \text{Type}_j) \to A \to \text{Type}_j) \\
(\text{iso\_rel} \ A \to \text{eq} \ A \to \text{iso}) \\
(\text{Proper}_{i \, j} \ A) \]

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

Actually, a non-polymorphic \textit{crelation}(A : \text{Type}_i) := A \to \to \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 $\mathcal{C} = ? : \mathcal{C}[x] \to \mathcal{C}[y]$
- Isomorphism of types
- Computationally relevant relations, e.g. CoRN’s appartness relation on reals.
- Hom-types of categories which are not $\text{Prop}$-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

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

M. Sozeau - Proof-Relevant Rewriting Strategies
try $s$ = $s ||$ id

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

repeat $s$ = $s ;$ any $s$

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

topdown $s$ = fix $td$.($s ||$ (progress (subterms $td$))) ; try $td$

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

outermost $s$ = fix $o$.($s ||$ (subterm $o$))
An example

Suppose the theory of monoids on T.
A goal: $x \ y : T \vdash x \cdot ((\epsilon \cdot y) \cdot \epsilon)$.

- 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 $\epsilon \cdot y$ to $y$ and directly after, $y \cdot \epsilon$ 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 $E_{simpl}$)

- Avoid mixing of rewrite with Ltac constructs, e.g.:
  
  (rewrite $l_1$ || ... || progress rewrite $l_n$) becomes
  
  rewrite_strat ($l_1$ || ... || progress $l_n$) which traverses the term just once.

- Another common pattern:

  ```plaintext
  match goal with
  |- context [t] => rewrite l
  end
  =
  rewrite_strat (topdown (pattern t; term l))
  ```
Future work

- Debug & release
- Benchmarks
The End