Proof-Relevant Rewriting Strategies (WIP)

Matthieu Sozeau – Inria Paris & PPS

6th Coq Workshop
July 17th 2014
Vienna, Austria

Generalized Rewriting

- **Equational reasoning** `x = y |- x + 1 ==> y + 1`
- **Logical reasoning** `x <-> y |- (x /\ y) ==> (x /\ x)`
- **Rewriting** `y ~> z |- x ~> y ==> x ~> z`
- **Abstract data types, quotients/setoids**
  `s, t : list, x =set y |- union x y =set x`
  `==> union x x =set x`

Moving from substitution to congruence.

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

Rewriting in Type Theory

Moving from substitution to congruence.

- Built-in substitution: Leibniz equality.
  $\Pi\ A\ (P : A \to \texttt{Type})\ (x\ y : A),\ P\ x \to x = y \to P\ y.$
  - ✓ 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.
  $\Pi\ A\ (P : A \to \texttt{Type})\ (x\ y : A),\ P\ x \to x = y \to P\ y.$
  - ✓ 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
  Proper $(\mathrm{iff} \mathrel{++>} \mathrm{iff})\ \mathrm{not} \triangleq \Pi\ P\ Q,\ P \leftrightarrow Q \to \neg\ P \leftrightarrow \neg\ Q$
- Multiple signatures for a given constant
  Proper $(\mathrm{impl} \longrightarrow \mathrm{impl})\ \mathrm{not}$

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

- Generalized to any relation
  $\mathsf{Proper}\ (\mathrm{iff} \Longrightarrow \mathrm{iff})\ \mathrm{not} \triangleq \Pi\ P\ Q,\ P \leftrightarrow Q \rightarrow \neg\ P \leftrightarrow \neg\ Q$
- Multiple signatures for a given constant
  $\mathsf{Proper}\ (\mathrm{impl} \longrightarrow \mathrm{impl})\ \mathrm{not}$

Requires proof search:

- Heuristic in NuPRL based on subrelations ($\mathrm{impl} \subset \mathrm{iff}$)
- Complete procedure in Coq.

Both are monolithic algorithms with a primitive notion of signature: a list of atomic relations (with variance).

A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\ A : \mathtt{Type}, (A \to \mathtt{Prop}) \to \mathtt{Prop}$

  $\Pi\ A, \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (\mathtt{@}\mathrm{all}\ A)$

A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\, A : \texttt{Type}, (A \rightarrow \texttt{Prop}) \rightarrow \texttt{Prop}$

  $\Pi\, A, \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (\texttt{@}\mathrm{all}\ A)$

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

  $\Pi\, A\ B\ C\ R_0\ R_1\ R_2,$
  $\quad \mathsf{Proper}\ ((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
  $\qquad (\texttt{@}\mathrm{compose}\ A\ B\ C)$

A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\, A : \texttt{Type}, (A \to \texttt{Prop}) \to \texttt{Prop}$

  $\Pi\, A,\ \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (@\mathrm{all}\ A)$

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

  $\Pi\, A\ B\ C\ R_0\ R_1\ R_2,$
  $\quad \mathsf{Proper}\ ((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
  $\qquad (@\mathrm{compose}\ A\ B\ C)$

- Generic morphism declarations.

A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\, A : \mathtt{Type},\ (A \to \mathtt{Prop}) \to \mathtt{Prop}$

  $\Pi\ A,\ \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (@\mathrm{all}\ A)$

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

  $\Pi\ A\ B\ C\ R_0\ R_1\ R_2,$
  $\quad \mathsf{Proper}\ ((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
  $\qquad (@\mathrm{compose}\ A\ B\ C)$

- Generic morphism declarations.
- Support for subrelations, quotienting the signatures.

A New Look at Generalized Rewriting

Sozeau [JFR 2009]

- Extensible signatures (shallow embedding)

  $\mathrm{all} : \forall\ A : \texttt{Type},\ (A \to \texttt{Prop}) \to \texttt{Prop}$

  $\Pi\ A,\ \mathsf{Proper}\ (\mathrm{pointwise\_relation}\ A\ \mathrm{iff} \Longrightarrow \mathrm{iff})\ (\texttt{@}\mathrm{all}\ A)$

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

  $\Pi\ A\ B\ C\ R_0\ R_1\ R_2,$
  $\quad \mathsf{Proper}\ ((R_1 \Longrightarrow R_2) \Longrightarrow (R_0 \Longrightarrow R_1) \Longrightarrow (R_0 \Longrightarrow R_2))$
  $\quad\quad (\texttt{@}\mathrm{compose}\ A\ B\ C)$

- Generic morphism declarations.
- Support for subrelations, quotienting the signatures.
- Rewriting on operators/functions, parallel rewrites. . .

Proper

```
Class Proper {A} (R : relation A) (m : A) : Prop :=
  proper : R m m.

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

Proper

```
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).
```

Proper

```
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).
```

Proper

```
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.
```

Outline

Impredicativity helped

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

$$\mathsf{Proper} : \Pi A : \texttt{Type}_i, (A \to A \to \texttt{Type}_j) \to A \to \texttt{Type}_j.$$

Need to show, under $A : \texttt{Type}_i$:

$$\begin{array}{ll} \mathsf{Proper} & ((A \to A \to \texttt{Type}_j) \to A \to \texttt{Type}_j) \\ & (\mathrm{iso\_rel}\ A \longrightarrow \mathrm{eq}\ A \longrightarrow \mathrm{iso}) \\ & (\mathsf{Proper}\ A) \end{array}$$

Inconsistency: $\texttt{Type}_{\mathsf{max}(i,j+1)} \not\leq \texttt{Type}_i$

Universe Polymorphism

With full universe polymorphism (Sozeau & Tabareau [ITP'14]):

$$\mathsf{Proper}_{i\,j} : \Pi A : \mathtt{Type}_i, (A \to A \to \mathtt{Type}_j) \to A \to \mathtt{Type}_j$$

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

$$\begin{array}{ll} \mathsf{Proper}_{i'\,j'} & ((A \to A \to \mathtt{Type}_j) \to A \to \mathtt{Type}_j) \\ & (\mathrm{iso\_rel}\ A \longrightarrow \mathrm{eq}\ A \longrightarrow \mathrm{iso}) \\ & (\mathsf{Proper}_{i\,j}\ A) \end{array}$$

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

Actually, $\mathrm{crelation}(A : \mathtt{Type}_i) := A \to A \to \mathtt{Type}_j$ is already problematic: no relation equivalence or subrelation definition possible.

Computational relations

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.

Outline

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.

Rewriting strategies

- 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` $strategy$ $(\texttt{in}\ t)?$

Rewriting strategies

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

Derived strategies

$$
\begin{array}{lcl}
\texttt{try}\ s & = & s \mid\mid \texttt{id} \\
\texttt{any}\ s & = & \texttt{fix}\ u.\texttt{try}\ (s\ \texttt{;}\ u) \\
\texttt{repeat}\ s & = & s\ \texttt{;}\ \texttt{any}\ s \\
\texttt{bottomup}\ s & = & \texttt{fix}\ bu.((\texttt{progress}\ (\texttt{subterms}\ bu)) \mid\mid s)\ \texttt{;}\ \texttt{try}\ bu \\
\texttt{topdown}\ s & = & \texttt{fix}\ td.(s \mid\mid (\texttt{progress}\ (\texttt{subterms}\ td)))\ \texttt{;}\ \texttt{try}\ td \\
\texttt{innermost}\ s & = & \texttt{fix}\ i.((\texttt{subterm}\ i) \mid\mid s) \\
\texttt{outermost}\ s & = & \texttt{fix}\ o.(s \mid\mid (\texttt{subterm}\ o))
\end{array}
$$

An example

Suppose the theory of monoids on `T`.
A goal: `x y : T |- x` $\bullet$ `((`ϵ $\bullet$ `y)` $\bullet$ ϵ`)`.

- `autorewrite with monoids` will do two rewrites with both unit laws, the proof term will be roughtly twice the goal size.
- `rewrite_strat (topdown (repeat (hints monoids)))` will first rewrite ϵ $\bullet$ `y` to `y` and directly after, `y` $\bullet$ ϵ 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.: (`rewrite` l_1 || ... || `progress rewrite` l_n) becomes `rewrite_strat` (l_1 || ... || `progress` l_n) which traverses the term just once.
- Another common pattern:

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

Future work

- Debug & release
- Benchmarks

The End