Polishing Up the Church–Rosser Theorem

Randy Pollack

Version of October 26, 2011
Let \( R \in Rel2 \), the class of binary relations.

the **transitive-reflexive closure** of \( R \), written \( R^* \), is defined inductively by

\[
\begin{align*}
\frac{aRb}{aR^*b} & \quad \text{*-base} \\
\frac{aR^*a}{aR^*b} & \quad \text{-refl} \\
\frac{aR^*b \quad bR^*c}{aR^*c} & \quad \text{-trans}.
\end{align*}
\]

\( R \) has the **diamond property**, \( dp(R) \), iff

\[
\forall a, b, c . \ aRb \land aRc \Rightarrow \exists d . \ bRd \land cRd.
\]

\( R \) is **confluent** iff \( dp(R^*) \).
Recall some $\lambda$-calculus

- $x, y, z, \ldots$, range over variables.
- $\lambda$-terms are ranged over by $a, b, c$:
  \[ a ::= x \mid \lambda x.a \mid ab. \]

We are always speaking up to $\alpha$-conversion.

- One-step $\beta$-reduction is defined by:
  \begin{align*}
  (\lambda u.b) a & > [a/u]b \quad (\beta) \\
  a > a' & \quad (\xi) \\
  a > a' & \quad (appl) \\
  b > b' & \quad (appr)
  \end{align*}

- Substitution lemma.
  If $a >^* b$, then $[a/x]c >^* [b/x]c$ and $[c/x]a >^* [c/x]b$.
- A term, $a$, is in $>^*$-normal form iff $a$ has no $>-reductions.$
Church-Rosser (CR) theorem

- The CR theorem states that $\Rightarrow$ is confluent, i.e. $dp(\Rightarrow^*)$.
- The same lemma holds for combinatory reduction for $s, k$ combinators. The same proof idea works.
- **corollary:** normal forms are unique: if $a \Rightarrow^* b$, $a \Rightarrow^* c$, and $b$, $c$ are both in normal form, then $b = c$ (up to $\alpha$-conversion).
  - **proof** By diamond property, $b$, $c$ reduce to a common term, $d$. But $b$, $c$ are both in normal form, so $b = d = c$.
- CR does *not* say that every term has a normal form, or that if one reduction sequence reaches a normal form then every reduction sequence reaches a normal form.
Strip lemmas

For any $\rightarrow \in \text{Rel2}$, $dp(\rightarrow) \Rightarrow dp(\rightarrow^*)$ by the diagram chase:

![Diagram]

This is really double induction:

- first along the top, showing that every rectangle of height 1 commutes (called the *strip lemma*),
- then along the side, showing that every rectangle commutes.

See my paper to get same result with a single induction.

Thus, if we had $dp(\rightarrow^*)$, we would be finished; but that is not the case. **Two things go wrong.**
What is wrong with >?

(1) > can forget parts of a term, but is not reflexive

\[(\lambda x. y) ((\lambda x. x) z)\]

\[(\lambda x. y) z\]

\[y\]

\[y\]
What is wrong with $\Rightarrow$?

(2) $\Rightarrow$ can copy parts of a term, but is not parallel

$$(\lambda x.xx)((\lambda x.y)z)$$

$$(\lambda x.y)z \quad (\lambda x.y)z$$

$$(\lambda x.xx)y$$

$y \quad y \quad y$$
An Outline of the Proof

The idea of Tait and Martin-Löf: define a relation of parallel reduction, \( \gg \), that is both reflexive and parallel.

1. The subtle part is showing \( dp(\gg) \).
   - I present an improvement (due to Takahashi) of the Tait–Martin-Löf proof.

2. The easy part is showing \( dp(\gg) \) implies \( dp(\gg^*) \).
   - This is the strip lemma we saw above.

3. Showing \( a \gg^* b \) iff \( a >^* b \) (hence \( dp(\succ^*) \), our goal).
   - This is usually considered trivial, but in fact the names of variables are problematic.
   - We skip the problematic details.
Parallel Reduction, $\gg$

- **pr-refl**
  - $x \gg x$

- **pr-β**
  - $a \gg a' \quad b \gg b' \\ 
  (\lambda u.a) b \gg [b'/u]a' \\
  \hline
  \quad a \gg a' \\
  \lambda u.a \gg \lambda u.a' \\
  \hline
  \quad a \gg a' \quad b \gg b' \\
  a \ b \gg a' \ b'$

- **pr-app**
  - $a \gg a' \quad b \gg b' \\
  \quad a \ b \gg a' \ b'$

- $\gg$ is reflexive by rule **pr-refl**.

- General reflexivity, $a \gg a$, is derivable.

- $\gg$ is parallel: rules **pr-β** and **pr-app** allow reduction in both subterms.

- Non-deterministic choice of which rule to apply to a redex, **pr-β** or **pr-app**.
Intuition about parallel reduction (\(\gg\))

- Any 2 redexes in a term are either disjoint, or one is contained in the other.
  - All redexes are subterms, and subterms have this property.
  - This holds for combinator terms as well as \(\lambda\)-terms.

- We can unambiguously mark each redex in a term, say by putting a unique identifier on its outer application.

- \(\gg\) allows contracting any subset of the marked redexes
  - Contracting may discard some redexes, and may copy some redexes (copy the marks with the redexes).
  - Contracting may also create brand new unmarked redexes.
  - The redexes with a particular mark left after a \(\gg\)-step are called **residuals** of the original redex with that mark.
Intuition about the proof of CR

- To prove $dp(\rightarrow\rightarrow)$, we are given 2 reductions, $a \rightarrow\rightarrow b$, $a \rightarrow\rightarrow c$, contracting different sets of marked redexes, say $M_b$ and $M_c$.
- To complete the diamond, just contract the necessary marked redexes.
  - In $b$, (resp. $c$), contract any redexes from $M_c$ (resp. $M_b$) that are left.
  - Any redexes not in $M_b$ or $M_c$ can be ignored.
  - Any new (unmarked) redexes can be ignored.
- At the end the same set of redexes will have been contracted along both reduction paths, so they will end at the same term.
**dp(≫): Parallel Reduction has the diamond property**

- We could actually mark the redexes (Huet 1994) . . .
- usual proof keeps track implicitly of which redexes are contracted.

**lemma (CR):** \( \forall a, b, c . a \gg b \land a \gg c \Rightarrow \exists d . b \gg d \land c \gg d. \)

**proof** By “induction on the structure of \( a \)” (Shankar 1988):

- \( a \) is a variable (trivial)
- \( a \) is a lambda (easy)
- \( a \) is an application; 5 subcases
  1. \( a \) is not a redex
  2. \( a \) is a redex, and is only contracted in the reduction \( a \gg b \)
  3. \( a \) is a redex, and is only contracted in the reduction \( a \gg c \)
  4. \( a \) is a redex, and is contracted in both reductions
  5. \( a \) is a redex, and is not contracted in either reduction
...proving \( dp(\gg) \)

For example in case 4, suppose \( a = (\lambda x. a_l) a_r \) and \( a \gg b \) (respectively \( a \gg c \)) by two instances of \textit{pr-beta}:

\[
\frac{a_l \gg a^b_l \quad a_r \gg a^b_r}{(\lambda x. a_l) a_r \gg [a^b_r/x]a^b_l} \quad \frac{a_l \gg a^c_l \quad a_r \gg a^c_r}{(\lambda x. a_l) a_r \gg [a^c_r/x]a^c_l}
\]

By the two induction hypotheses we have the diagrams:

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array{...
Another Proof of \( dp(\gg) \)

- By “simultaneous induction on the structure of the reductions” \( a \gg b \) and \( a \gg c \) (Pfenning 1992).
- The argument goes as above, but the two uses of IH diagrammed above are justified because the derivations of
  - \( a_l \gg a_l^b \) and \( a_l \gg a_l^c \)
  - respectively \( a_r \gg a_r^b \) and \( a_r \gg a_r^c \)

are subderivations of the given derivation pair, \( a \gg b \) and \( a \gg c \).
These proofs are too fine!

- They analyze what redexes are contracted in the two given reductions, $a \gg b$ and $a \gg c$;
- they close the diamond by contracting only the redexes that are necessary to bring $b$ and $c$ together.

Why be so careful?
- In $b$, contracting all the marked redexes left
  - without regard to what redexes were contracted in $a \gg c$
    will have the same effect as if we just contracted all the redexes in $a$ to start with.
- The same is true of $c$.
- This will close the diamond, although it may contract some redexes that were not necessary to do so.
- the bottom of the diamond will be the “maximum $\gg$-step from $a$”.
A Coarser Proof of \( dp(\ggg) \) (Takahashi)

- Taking the “maximum” parallel reduction step that contracts all redexes in \( a \), we can close any triangle

\[
\begin{array}{c}
\ggg \\
\cdot \\
\ggg \\
\end{array}
\longrightarrow
\begin{array}{c}
a \\
\cdot \\
d \\
\end{array}
\]

by contracting all the residuals in \( b \) of redexes in \( a \).

- Then we can complete any diamond by closing the left and right triangles independently.

Define a new relation, called **complete development**, \( \ggg \), that contracts all the redexes in a term.
Complete Developments

\[ \text{cd-var} \quad x \gg x \]

\[ \text{cd-}\beta \quad \frac{a \gg a'}{ \lambda u. a \gg [b'/u] a'} \]

\[ \text{cd-}\xi \quad \frac{a \gg a'}{\lambda u. a \gg \lambda u. a'} \]

\[ \text{cd-app} \quad \frac{a \gg a' \quad b \gg b'}{a b \gg a' b'} \quad a \text{ is not an abstraction} \]

\[ \gg \] is “the same as \(\gg\) but goes farther”:

- The non deterministic choice in \(\gg\) to use \(pr-\beta\) or \(pr-app\) on a redex is removed.
- \(\gg\) contracts every redex (but not newly created ones).
is the “maximum parallel reduction”

- **lemma** (≫ exists) Every term has a complete development:
  \[ \forall a \exists d . \ a \gg d . \]

- **proof** Easy structural induction on \( a \): in every case exactly one rule applies.

In fact complete development is unique, but we don’t need that fact.

- **lemma** (triangle) ≫ closes any ≫ triangle:
  \[ \forall a, d, b . \ a \gg d \land a \gg b \Rightarrow b \gg d . \]

\[ a \]
\[ \gg \]
\[ b \]
\[ \gg \]
\[ \gg \]
\[ \gg \]
\[ d \]
Proof that $a ≫ d \land a ≫ b \Rightarrow b ≫ d$

Structural induction on $a ≫ d$.

- Consider the case where $a = (\lambda u.a_l) a_r ≫ [a_r^d / u] a_l^d = d$
  
  because

  \[
  \frac{a_l ≫ a_l^d \quad a_r ≫ a_r^d}{(\lambda u.a_l) a_r ≫ [a_r^d / u] a_l^d} \text{ (cd-β)}
  \]

- Two subcases for the 2 possible $≫$-steps from $a = (\lambda u.a_l) a_r$:
  - First subcase: $(\lambda u.a_l) a_r$ reduces by $pr-β$:

    \[
    \frac{a_l ≫ a_l^b \quad a_r ≫ a_r^b}{(\lambda u.a_l) a_r ≫ [a_r^b / u] a_l^b} \quad \text{ (pr-β)}
    \]

  - By IH we have $a_l^b ≫ a_l^d$ and $a_r^b ≫ a_r^d$,
  - by substitution lemma, $[a_r^b / u] a_l^b ≫ [a_r^d / u] a_l^d$ as required.
... $a \gg d \land a \gg b \Rightarrow b \gg d$

... proof continued

- Still the case where $a = (\lambda u. a_l) a_r \gg [a_r^d/u]a_l^d = d$ because

$$
\begin{align*}
  a_l \gg a_l^d & \quad a_r \gg a_r^d \\
  \frac{}{(\lambda u. a_l) a_r \gg [a_r^d/u]a_l^d} \quad (cd-\beta)
\end{align*}
$$

- Second subcase: $(\lambda u. a_l) a_r$ reduces by $pr$-$app$:

$$
\begin{align*}
  a_l \gg a_l^b \\
  \frac{}{(\lambda u. a_l) a_r \gg (\lambda u. a_l^b) a_r^b} \quad pr$-$lda \\
  \frac{}{(\lambda u. a_l) a_r \gg (\lambda u. a_l^b) a_r^b} \quad pr$-$app
\end{align*}
$$

- By IH we have $a_l^b \gg a_l^d$ and $a_r^b \gg a_r^d$,
- by rule $pr$-$\beta$, $(\lambda u. a_l^b) a_r^b \gg [a_r^d/u]a_l^d$ as required.
Church–Rosser theorem

**Lemma** \( dp(\gg) \): \( a \gg b \land a \gg c \Rightarrow \exists d . b \gg d \land c \gg d. \)

**Proof** Let \( d \) be s.t. \( a \gg d \) (\( \gg \) exists). \( b \gg d \) and \( c \gg d \) by (triangle).

This proof is **coarser** than the standard one:
- It treats less cases, by using a deterministic relation \( \gg \) instead of figuring out which redexes must be contracted by \( \gg \).
- It produces a **worse program to compute** \( d \), contracting more redexes than necessary.

**Corollary** \( dp(\gg^*) \)

**Proof** By a strip lemma diagram chase.

**Corollary** \( a \gg^* b \iff a \succ^* b \); hence \( dp(\succ^*) \).

**Proof** (\( \leftarrow \)) \( a \succ b \Rightarrow a \gg b \) is trivial, as every \( \succ \)-step is also a \( \gg \)-step.

(\( \rightarrow \)) \( a \gg b \Rightarrow a \succ^* b \) is proved by induction on the derivation of \( a \gg b \).
More about $\gg$

- $\gg$ is not just a reduction relation, it is deterministic, i.e. $\gg$ is a strategy.
- **Lemma** $\gg^*$ is cofinal for $\ast$, i.e.

\[
\forall a, b . \ a \ast b \Rightarrow \exists d . \ a \gg^* d \land b \ast d
\]

- **Proof** Easy from what we have proved above.

- **Lemma** $\gg^*$ is normalizing: if $a$ has a normal form, then $\gg^*$ deterministically finds it.
  - **Proof** If $a_n$ is the normal form of $a$, then $a \ast a_n$, so for some $d$, $a \gg^* d$ and $a_n \ast d$. But $a_n$ is normal, so $a = d$.

- $\gg^*$ is an easier normalizing strategy to reason about than other such strategies (call-by-need, call-by-name, ...)
Conclusion

- Very well known proofs can be made more beautiful.
- Look for alternative inductive definitions that are easier to reason with.
  - In this search, try to eliminate unnecessary non-determinism in definitions.
- For program extraction must pay attention to the algorithmic content of proofs.