Confluence of the Lambda Calculus with Left-Linear Algebraic Rewriting *

Fritz Müller
email: (λx.muellerxcs.uni-sb.de)@
URL: rw4.cs.uni-sb.de/~mueller/lconfluence.ps.gz

1992

Abstract

We consider the untyped λ-calculus with β-reduction and algebraic rewrite rules for some constants. The rules must be left-linear and must not contain applications of variables in their left sides. We prove that the combined reduction relation is confluent (Church-Rosser) if the algebraic rewrite system alone is confluent. This result also holds when reduction is restricted to a subset of terms with suitable closure conditions. (This subset may be a set of well-typed terms.)

Keywords: lambda calculus, term rewriting system, confluence, Church-Rosser, parallel conditional

1 Motivating example

We introduce an example to motivate the general result of section 2: Consider a recursively typed λ-calculus [CC91, Cos89] with the type forming constructors + (separated sum) and $\rightarrow$ (function space), among others. The terms of the programming language are typed λ-expressions with higher order constants for constructing and destructing terms of non-functional type. In our case we consider the constructors $\mathsf{0}_{\sigma,\tau} : \sigma \rightarrow (\sigma+\tau)$ and $\mathsf{1}_{\sigma,\tau} : \tau \rightarrow (\sigma+\tau)$ for elements of sum type ("inleft", "inright") and a *parallel* conditional $\mathsf{pcond}_{\sigma,\tau,\rho} : (\sigma+\tau) \rightarrow \rho \rightarrow \rho \rightarrow \rho$. (We will omit the type subscripts and regard the operators as single and untyped in our rewrite system.)

We give a reduction relation, $\longrightarrow$, on terms by the usual β-reduction and the following algebraic rewrite rules:

(pcond 0)	$\mathsf{pcond}\ (\mathsf{0}\ \mathrm{x})\ \mathrm{y}\ \mathrm{z}$	$\longrightarrow$	y
(pcond 1)	$\mathsf{pcond}\ (\mathsf{1}\ \mathrm{x})\ \mathrm{y}\ \mathrm{z}$	$\longrightarrow$	z
(pcond+0)	$\mathsf{pcond}\ \mathrm{x}\ (\mathsf{0}\ \mathrm{y})\ (\mathsf{0}\ \mathrm{z})$	$\longrightarrow$	$\mathsf{0}\ (\mathsf{pcond}\ \mathrm{x}\ \mathrm{y}\ \mathrm{z})$
(pcond+1)	$\mathsf{pcond}\ \mathrm{x}\ (\mathsf{1}\ \mathrm{y})\ (\mathsf{1}\ \mathrm{z})$	$\longrightarrow$	$\mathsf{1}\ (\mathsf{pcond}\ \mathrm{x}\ \mathrm{y}\ \mathrm{z})$
(pcond $\rightarrow$)	$\mathsf{pcond}\ \mathrm{x}\ \mathrm{y}\ \mathrm{z}\ \mathrm{w}$	$\longrightarrow$	$\mathsf{pcond}\ \mathrm{x}\ (\mathrm{y}\ \mathrm{w})\ (\mathrm{z}\ \mathrm{w})$

*published in Information Processing Letters 41(1992)293–299, please use the electronic version

There may be also other rules, e.g. for the product type or for the usual sequential conditional.

How can we prove the confluence (Church-Rosser property) of this system? I do not know of any theorem in the literature that provides an easy check. Regard the peculiarities here:

- The left sides of the rewrite rules overlap, they form critical pairs. The rules are left-linear, but not "non-ambiguous" in the sense of [Klo80, p. 130], so Klop's results on the confluence of "regular combinatory reduction systems" cannot be applied.

  Overlapping reduction rules are needed for the syntactical definition of several "fairly parallel" functions, see also the parallel or and the classical parallel $\mathsf{if\_then\_else}$ of [Plo77]. The definability of these functions is in turn necessary for the full abstractness of the usual denotational semantics.

- Although we have a typed system, it is not normalizing. (There are non-terminating programs, e.g. the fixed point combinator is definable for every type.) Therefore all the results on the confluence of λ-calculus + algebraic rewriting that rely on strong normalization [BT88, BTG89, Bar90, Dou91, JO91] do not apply here.

  But well-typing is also essential for the confluence of our system:
  The term $\mathsf{pcond\ x\ (0\ y)\ (0\ z)\ w}$ by (pcond $\rightarrow$) reduces to $\mathsf{pcond\ x\ (0\ y\ w)\ (0\ z\ w)}$ and by (pcond+0) to $\mathsf{0\ (pcond\ x\ y\ z)\ w}$, and these two terms have no common reduct. But $\mathsf{pcond\ x\ (0\ y)\ (0\ z)\ w}$ is not typable, as $\mathsf{(0\ y)}$ is not of function type.

- The terms of our algebraic rewrite system are constructed of application as the only non-0-ary symbol (it is invisible) and of 0-ary constants and variables. (These arities are meant in the (low level) sense of algebraic terms, not in the type sense!) We will exclude terms with an application of a variable (xM) only on the left side of rules, but not on the right side. But the prevailing concept of algebraic rewriting + λ-calculus defines algebraic terms as applications of constants to (sequences of) algebraic terms, so it excludes (xM) entirely, e.g. [Dou91]. This would exclude e.g. our rule (pcond $\rightarrow$). In order to make the distinction clear, we use the usual notion "algebraic" only in the title and replace it by our "applicative" (terms, term rewriting systems) in the following section 2.

In section 2 we prove the confluence of the λ-calculus with left-linear, not variable-applying, confluent rules. For our example it remains to show the confluence of the rewrite rules alone. This can be done by proving the termination, e.g. by mapping each term to a number that decreases in every reduction step, and the convergence of all critical pairs.
Our notation $x \equiv y$ means: x is defined to be y.

2 Confluence of the λ-calculus with left-linear applicative rewriting

2.1 Terms and occurrences

We fix a finite or denumerable set $\mathcal{C}$ of higher-order *constants* and a denumerable set $\mathcal{V}$ of *variables*, with $\mathcal{C} \cap \mathcal{V} = \emptyset$. Variables are denoted by x, y. (λ-)*terms* are built from constants and variables by *application* MN and (λ-)*abstraction* $\lambda x.M$. They are considered equal modulo renaming of bound variables (α-conversion). Λ is the set of all terms. The set $\mathcal{A}$ of *applicative terms* is the set of all terms without any abstraction. The function *Var* maps an applicative term to the set of its variables.

Occurrences of subterms are (as usual) sequences of $0, 1, \lambda$, denoted by u, v, w, z. $v \leq w$ iff v is a prefix of w. v, w are *disjoint* ($v \mid w$) iff neither $v \leq w$ nor $w \leq v$. We say u *is in* term M if M has a subterm at u:
$M/\varepsilon \equiv M$, $(MN)/0w \equiv M/w$, $(MN)/1w \equiv N/w$, $(\lambda x.M)/\lambda w \equiv M/w$.

Replacement at an occurrence: $M[\varepsilon{\leftarrow}N] \equiv N$, $(MN)[0w{\leftarrow}P] \equiv M[w{\leftarrow}P]N$, $(MN)[1w{\leftarrow}P] \equiv MN[w{\leftarrow}P]$, $(\lambda x.M)[\lambda w{\leftarrow}P] \equiv \lambda x.M[w{\leftarrow}P]$.

If (w_i), $1 \leq i \leq n$, is a sequence of pairwise disjoint occurrences and (N_i) a corresponding sequence of terms, then $M[\vec{w_i}{\leftarrow}\vec{N_i}] \equiv M[w_1{\leftarrow}N_1]\ldots[w_n{\leftarrow}N_n]$.

2.2 Applicative term rewriting system

An *applicative term rewriting system (ATRS)* is any set of pairs $\langle L{\rightarrow}R\rangle$ of applicative terms, where L is no variable and $\mathit{Var}\,R \subseteq \mathit{Var}\,L$. Each pair is called a rule of the ATRS. (Note that applicative terms are built only from constants, variables, and application.)

A *substitution* is a map $\sigma : \mathcal{V} \rightarrow \Lambda$. It naturally extends to applicative terms.
The ATRS together with β-reduction defines a one step reduction relation, $\longrightarrow$, on Λ; it is the least relation satisfying:

(β) $(\lambda x.M)N \longrightarrow M[x{:=}N]$ for any terms M, N and variable x, where $M[x{:=}N]$ is the substitution of N for the free occurrences of x in M, with appropriate renaming of bound variables,

($\langle L{\rightarrow}R\rangle$) for all rules $\langle L{\rightarrow}R\rangle$: $\sigma L \longrightarrow \sigma R$ for any substitution σ,

(app) $M \longrightarrow M' \Longrightarrow MN \longrightarrow M'N$,
$N \longrightarrow N' \Longrightarrow MN \longrightarrow MN'$,

(λ) $M \longrightarrow M' \Longrightarrow \lambda x.M \longrightarrow \lambda x.M'$.

$\longrightarrow$ reduces applicative terms to applicative terms. The restriction of $\longrightarrow$ to $\mathcal{A}$ is the least relation on $\mathcal{A}$ satisfying ($\langle L{\rightarrow}R\rangle$) for all rules and the two (app) conditions.

Let $M \longrightarrow N$ hold. Then there is an occurrence u in M where either M/u is of the form $(\lambda x.M_1)M_2$ and $N = M[u{\leftarrow}M_1[x{:=}M_2]]$ or M/u is of the form σL and $N = M[u{\leftarrow}\sigma R]$ for the rule $\langle L{\rightarrow}R\rangle$. $M \longrightarrow N$ together with such u and the used rule is said to be a *reduction step*. M/u is called the *redex* of the reduction step, u is called the occurrence of the redex resp. of the reduction step, and the reduction *is at u with rule* (β) resp. $\langle L{\rightarrow}R\rangle$.

We define descendants of occurrences for the second case $\langle L{\rightarrow}R\rangle$: Let w be an occurrence in M beneath a variable of L, i.e. $w = uvz$ with $L/v = y \in \mathcal{V}$. Then the *descendants* of w after the reduction step $M \longrightarrow N$ at u with rule $\langle L{\rightarrow}R\rangle$ are all $uv'z$ with $R/v' = y$. If $w|u$ then w is the only *descendant* of w. (The other cases for w do not occur in the following.)

An ATRS is called *left-linear* iff the left side of each rule is linear, i.e. every variable has at most one occurrence in the left side.

An ATRS is called *variable-applying* iff there is a left side with a subterm of the form (xM), where $x \in \mathcal{V}$.

We will impose the restriction that our ATRS is left-linear and *not* variable-applying. Left-linearity is motivated by non-confluent examples of the λ-calculus with non-left-linear rewrite rules, e.g. surjective pairing [Klo80, III.1]. This restriction seems to be reasonable for our intended use of ATRSs as definitions of deterministic programming languages. The detection of a non-linear left side cannot be seen as an atomic operational step.

The not-variable-applying condition is motivated e.g. by the ATRS with the single rule $\mathsf{x}\ \mathsf{y} \longrightarrow \mathsf{x}$. We get the following non-converging reductions: $\lambda x.x \longleftarrow (\lambda x.x)y \longrightarrow y$. "Not variable-applying" ensures that there is no overlap between the left side of some rule of the ATRS and the left side of the β-reduction rule. Then the ATRS can be said to be "orthogonal" to β-reduction, there are no "critical pairs" between the two reduction systems. We will prove that in this case, together with left-linearity, the two reduction relations commute.

Remarks: [RV80, Proposition 10] and [Toy88, Corollary 3.1] prove a similar result: Two left-linear term rewriting systems commute if they do not overlap with each other. The not-variable-applying condition appears also in [Nip91, Lemma 5.1]: The β-reduction rule and the applicative rules are both formalized as rules of a single higher-order rewrite system, so higher-order critical pairs between them can be defined. Due to a general critical pair lemma, the *local* confluence of the combined reduction system follows from the local confluence of the applicative rules and the not-variable-applying condition. Left-linearity is not needed there. Note that we prove full confluence.

2.3 $\rightarrow$-closed sets of terms

To obtain an abstract notion of "well-typing" we must restrict the reduction to certain subsets of Λ. These can be rather arbitrary, but must fulfill three closure conditions with respect to our reduction $\longrightarrow$:

A set $T \subseteq \Lambda$ is called *$\rightarrow$-closed* iff for every $M \in T$ the following holds:

1) $M \longrightarrow M' \Longrightarrow M' \in T$.

2) Every subterm of M is in T.

3) For every occurrence u of an abstraction in M, $M/u = \lambda \ldots$,
there is a variable x not occurring in M with $M[u{\leftarrow}x] \in T$.

The set of applicative terms of T is also $\rightarrow$-closed. In our example T would be the set of typable terms; its closedness is a consequence of the "subject reduction property".

Remark: [Dou91] uses two more closure conditions concerning strong normalization, as he does not assume left linearity. Condition 3 is missing in that work. Our reason for it is the weaker premise of our confluence theorem: While [Dou91] assumes confluence on all algebraic terms, we assume confluence on algebraic, resp. applicative terms of $\underline{T}$ only. Condition 3 ensures that there are enough applicative terms in T.

2.4 Confluence Theorem

$\stackrel{*}{\longrightarrow}$ means the reflexive, transitive closure of the relation $\longrightarrow$.
$\longrightarrow$ is called *confluent* (Church-Rosser) on the $\rightarrow$-closed set T iff for all terms $M \in T$ with $N \stackrel{*}{\longleftarrow} M \stackrel{*}{\longrightarrow} P$ there is Q with $N \stackrel{*}{\longrightarrow} Q \stackrel{*}{\longleftarrow} P$. (These terms are all in T because of $\rightarrow$-closedness of T.)

Main theorem: *For every left-linear, not variable-applying ATRS with reduction relation $\longrightarrow$ and every $\rightarrow$-closed set T:*
$\longrightarrow$ is confluent on the applicative terms of $T \iff \longrightarrow$ is confluent on T.

Proof: The direction $\Longleftarrow$ is trivial.
For showing $\Longrightarrow$, we split $\longrightarrow$ into two parts: $\underset{\beta}{\longrightarrow}$ is the least relation on Λ satisfying (β), (app) and (λ). $\underset{A}{\longrightarrow}$ is the least relation on Λ satisfying ($\langle L{\rightarrow}R\rangle$) for all rules, (app) and (λ). We have $\longrightarrow \;=\; \underset{\beta}{\longrightarrow} \cup \underset{A}{\longrightarrow}$. $\longrightarrow$ is confluent on the applicative terms of T iff $\underset{A}{\longrightarrow}$ is confluent on the applicative terms of T.
$\underset{\beta}{\longrightarrow}$ is known to be confluent on all terms [Bar84, p. 59]. After two simple lemmas we will prove the confluence of $\underset{A}{\longrightarrow}$ on T (Lemma 2.4.3), and then the commutativity of $\underset{\beta}{\longrightarrow}$ and $\underset{A}{\longrightarrow}$ (Lemma 2.4.4). The assumptions "left-linear" and "not variable-applying" for the ATRS will be used only in the proof of the last Lemma 2.4.4.

Lemma 2.4.1 *If $M \underset{A}{\longrightarrow} M'$ then $M[x{:=}N] \underset{A}{\longrightarrow} M'[x{:=}N]$.*

Proof: The reduction step of the consequence is at the same occurrence, with the same rule $\langle L{\rightarrow}R\rangle$. We have to show $(\sigma L)[x{:=}N] \underset{A}{\longrightarrow} (\sigma R)[x{:=}N]$ for any σ.
Let σ' be the substitution $y \mapsto (\sigma y)[x{:=}N]$. Then:

$$(\sigma L)[x{:=}N] = \sigma' L \underset{A}{\longrightarrow} \sigma' R = (\sigma R)[x{:=}N]$$

Lemma 2.4.2 *If $N \underset{A}{\longrightarrow} N'$ then $M[x{:=}N] \underset{A}{\stackrel{*}{\longrightarrow}} M[x{:=}N']$.*

Proof: If the reduction step $N \underset{A}{\longrightarrow} N'$ is at u with rule $\langle L{\rightarrow}R\rangle$, we make a reduction with rule $\langle L{\rightarrow}R\rangle$ at all occurrences vu in $M[x{:=}N]$, where v is any free occurrence of x in M.

Lemma 2.4.3 $\xrightarrow[A]{}$ *is confluent on* T.
(This lemma does not use the assumptions "left-linear" and "not variable-applying" for the ATRS, but all the conditions of $\rightarrow$*-closedness of* T*.)*

Proof: We prove for any $M \in T$:
If $N \xleftarrow[A]{*} M \xrightarrow[A]{*} P$ then there is Q with $N \xrightarrow[A]{*} Q \xleftarrow[A]{*} P$, under the assumption of this property for all proper subterms of M (induction on M). (The subterms of M are all in T.)
Case 1: M is of the form $\lambda x.M'$. Then $N = \lambda x.N'$, $P = \lambda x.P'$ with $N' \xleftarrow[A]{*} M' \xrightarrow[A]{*} P'$. By the induction hypothesis there is Q' with $N' \xrightarrow[A]{*} Q' \xleftarrow[A]{*} P'$. We choose $Q \equiv \lambda x.Q'$.
Case 2: M is no abstraction. Let (u_i), $1 \leq i \leq m$ be a sequence of the outermost, i.e. minimal, occurrences of abstractions in M. The u_i's are pairwise disjoint. Let (x_i), $1 \leq i \leq m$, be a sequence of pairwise different variables, each one different from any variable in M, that can be used to replace the abstractions at u_i according to condition 3 of $\rightarrow$-closed sets. $M' \equiv M[\vec{u_i} \leftarrow \vec{x_i}]$ is an applicative term of T with $M = M'[x_1 := M/u_1] \ldots [x_m := M/u_m]$. We proceed with the following property of the reduction $M \xrightarrow[A]{*} N$:

Proposition: For every reduction $M \xrightarrow[A]{*} N$ there is an (applicative) term N' with

$$M' \xrightarrow[A]{*} N', \quad N = N'[\vec{v_j^1} \leftarrow \vec{N_j^1}] \ldots [\vec{v_j^m} \leftarrow \vec{N_j^m}],$$

where (v_j^i) is a sequence of the occurrences of variable x_i in N' for every i, and (N_j^i) is a corresponding sequence of terms with $M/u_i \xrightarrow[A]{*} N_j^i$ for any i,j.
This means: The variables (x_i) keep track of the corresponding subterms of M during the reduction $M' \xrightarrow[A]{*} N'$. The reductions in M' are independent of the reductions made in subterms M/u_i.
Proof: The proposition can be proved by induction on the length of the reduction $M \xrightarrow[A]{*} N$: For length 0 we have $N = M$ of the required form, $N' = M'$. Now assume we have reached a term N of the required form and make the next reduction step $N \xrightarrow[A]{} L$ at w with rule $\langle S \rightarrow T \rangle$. There are two cases:
Case 1: w is an occurrence in N'. Then the left side of the rule lies entirely in N', because all the N_j^i are abstractions. So there is a reduction step $N' \xrightarrow[A]{} L'$ at w with rule $\langle S \rightarrow T \rangle$. Every v_j^i has a set D of descendants after the reduction step $N \xrightarrow[A]{} L$. v_j^i is replaced in the sequence for i by a sequence of the elements of D, and N_j^i is replaced by a corresponding multiple sequence of the same N_j^i. Then L is L' with the new replacements of variable occurrences.
Case 2: Some $v_j^i \leq w$. Then L is N' with the old replacement, except that N_j^i is exchanged with its reduct.
End of Proof Proposition

Now we have $N \xleftarrow[A]{*} M \xrightarrow[A]{*} P$ with N in the form of the proposition, and P in analogous form, namely there is P' with

$$M' \xrightarrow[A]{*} P', \quad P = P'[\vec{w_j^1} \leftarrow \vec{P_j^1}] \ldots [\vec{w_j^m} \leftarrow \vec{P_j^m}],$$

(w_j^i) a sequence of occurrences of variable x_i in P', $M/u_i \xrightarrow[A]{*} P_j^i$.

We have $N' \xleftarrow[A]{*} M' \xrightarrow[A]{*} P'$. As $\xrightarrow[A]{}$ is confluent on applicative terms of T (and M' is one), there is Q' with $N' \xrightarrow[A]{*} Q' \xleftarrow[A]{*} P'$.
For $1 \leq i \leq m$: $M/u_i \xrightarrow[A]{*} N^i_j$ and $M/u_i \xrightarrow[A]{*} P^i_j$ for all j. M/u_i is a proper subterm of M (since M is no abstraction), so we can use the induction hypothesis to find a common reduct Q^i of all the N^i_j, P^i_j (for fixed i), i.e. $N^i_j \xrightarrow[A]{*} Q^i$, $P^i_j \xrightarrow[A]{*} Q^i$.

Then we choose $Q \equiv Q'[x_1{:=}Q^1] \ldots [x_m{:=}Q^m]$. We can reduce

$$N = N'[\vec{v^1_j} \leftarrow \vec{N^1_j}] \ldots [\vec{v^m_j} \leftarrow \vec{N^m_j}] \xrightarrow[A]{*} Q :$$

First we apply all the reductions $N^i_j \xrightarrow[A]{*} Q^i$ under every subterm occurrence v^i_j. As the (v^i_j) are *all* the occurrences of x_i in N', we get the term $N'[x_1{:=}Q^1] \ldots [x_m{:=}Q^m]$. This term reduces by $\xrightarrow[A]{*}$ and repeated application of Lemma 2.4.1 to Q.
Analogously, we can reduce $P \xrightarrow[A]{*} Q$.
End of Proof 2.4.3

Now $\xrightarrow[\beta]{}$ and $\xrightarrow[A]{}$ are confluent on T. We will show that $\xrightarrow[\beta]{}$ and $\xrightarrow[A]{}$ commute (on Λ), i.e. if $N \xleftarrow[\beta]{*} M \xrightarrow[A]{*} P$ then there is Q with $N \xrightarrow[A]{*} Q \xleftarrow[\beta]{*} P$. Then the confluence of $\longrightarrow$ on T follows from the Commutative Union Theorem of Hindley-Rosen [Ros73, 3.5] and from condition 1 of the $\rightarrow$-closedness of T.

For the proof of commutativity, we define a relation $\xrightarrow[\beta p]{}$ on Λ that makes multiple, parallel β-reductions in one step, by the following induction rules:

1) $M \xrightarrow[\beta p]{} M$

2) $(\lambda x.M)N \xrightarrow[\beta p]{} M[x{:=}N]$

3) $M \xrightarrow[\beta p]{} M'$, $N \xrightarrow[\beta p]{} N' \Longrightarrow MN \xrightarrow[\beta p]{} M'N'$

4) $M \xrightarrow[\beta p]{} M' \Longrightarrow \lambda x.M \xrightarrow[\beta p]{} \lambda x.M'$

$\xrightarrow[\beta p]{}$ makes a reduction on a disjoint set of redexes only. Contrary to the similar relation used in the Tait/Martin-Löf proof of the confluence of λ-calculus [Bar84, p. 60] there are no reductions inside M, N in rule 2. This is not necessary here, because the other, commuting relation $\xrightarrow[A]{}$ is "too weak" to cause nested β-reductions.

Lemma 2.4.4 *If $N \xleftarrow[\beta p]{} M \xrightarrow[A]{} P$ then there is Q with $N \xrightarrow[A]{*} Q \xleftarrow[\beta p]{} P$.*
(All terms are in Λ. This lemma uses the assumptions "left-linear" and "not variable-applying" for the ATRS.)

Proof: By induction on the derivation of $M \xrightarrow[\beta p]{} N$.
Case 1: $N = M$. We can choose $Q \equiv P$.
Case 2: $M \xrightarrow[\beta p]{} N$ is derived as $(\lambda x.M_1)M_2 \xrightarrow[\beta p]{} M_1[x{:=}M_2]$.
There is no rule $\langle L \rightarrow R \rangle$ applicable at occurrence ε in M, because the ATRS is not variable-applying and no left side is a variable. Therefore, $M \xrightarrow[A]{} P$ must be either of the form

$$(\lambda x.M_1)M_2 \xrightarrow[A]{} (\lambda x.M_1')M_2 \quad \text{with} \quad M_1 \xrightarrow[A]{} M_1'$$

or of the form

$$(\lambda x.M_1)M_2 \xrightarrow[A]{} (\lambda x.M_1)M_2' \quad \text{with} \quad M_2 \xrightarrow[A]{} M_2'.$$

By Lemma 2.4.1, we get in the first case:

$$M_1[x{:=}M_2] \xrightarrow[A]{} M_1'[x{:=}M_2] \xleftarrow[\beta p]{} (\lambda x.M_1')M_2,$$

by Lemma 2.4.2 in the second case:

$$M_1[x{:=}M_2] \xrightarrow[A]{*} M_1[x{:=}M_2'] \xleftarrow[\beta p]{} (\lambda x.M_1)M_2'.$$

Case 3: $M \xrightarrow[\beta p]{} N$ is derived as $M_1M_2 \xrightarrow[\beta p]{} N_1N_2$, as consequence of $M_1 \xrightarrow[\beta p]{} N_1$, $M_2 \xrightarrow[\beta p]{} N_2$.
Subcase 3.1: $M \xrightarrow[A]{} P$ is of the form $M_1M_2 \xrightarrow[A]{} P_1M_2$ with $M_1 \xrightarrow[A]{} P_1$.
We have $N_1 \xleftarrow[\beta p]{} M_1 \xrightarrow[A]{} P_1$, by induction hypothesis there is Q_1 with
$N_1 \xrightarrow[A]{*} Q_1 \xleftarrow[\beta p]{} P_1$. So $N_1N_2 \xrightarrow[A]{*} Q_1N_2 \xleftarrow[\beta p]{} P_1M_2$.
Subcase 3.2: $M \xrightarrow[A]{} P$ is of the form $M_1M_2 \xrightarrow[A]{} M_1P_2$ with $M_2 \xrightarrow[A]{} P_2$.
Analogous to subcase 3.1.
Subcase 3.3: $M \xrightarrow[A]{} P$ is of the form $\sigma L \xrightarrow[A]{} \sigma R$ for rule $\langle L{\to}R \rangle$, where (x_i), $1 \le i \le n$, are the variables of L and σ is the substitution that maps x_i to S_i.
Let us look at a single β-reduction in $M \xrightarrow[\beta p]{} N$ at occurrence u, thus M/u is of the form $(\lambda y.V)W$. If u is an occurrence in L it must be a variable occurrence, $L/u = x_j$ for some j. (Reason: Any other occurrence would imply L/u of the form (x_kW'), this is not possible since the ATRS is not variable-applying.) Hence in any case there is a variable occurrence v in L with $v \le u$. The x_i identify the variable occurrences uniquely, since the ATRS is left-linear.
Now there is a substitution σ' mapping x_i to S_i' with $S_i \xrightarrow[\beta p]{} S_i'$ and $N = \sigma' L$. We choose $Q \equiv \sigma' R$ and obtain

$$N = \sigma' L \xrightarrow[A]{} \sigma' R \xleftarrow[\beta p]{} \sigma R = P$$

.

Case 4: $M \xrightarrow[\beta p]{} N$ is derived as $\lambda x.M_1 \xrightarrow[\beta p]{} \lambda x.N_1$, as consequence of $M_1 \xrightarrow[\beta p]{} N_1$.
$M \xrightarrow[A]{} P$ must be of the form $\lambda x.M_1 \xrightarrow[A]{} \lambda x.P_1$ with $M_1 \xrightarrow[A]{} P_1$. By induction hypothesis there is Q_1 with $N_1 \xrightarrow[A]{*} Q_1 \xleftarrow[\beta p]{} P_1$. We choose $Q \equiv \lambda x.Q_1$.
End of Proof 2.4.4

From this lemma and the Commutativity Lemma of [Ros73, 3.6] it follows that $\xrightarrow[\beta]{}$ and $\xrightarrow[A]{}$ commute, because $\xrightarrow[\beta]{*} = \xrightarrow[\beta p]{*}$. As stated above, from this we can deduce the confluence of $\longrightarrow$ on T.
End of Proof Confluence Theorem 2.4

Acknowledgements: I thank Harald Ganzinger, Reinhold Heckmann, Helmut Seidl, Kurt Sieber and Reinhard Wilhelm for their interest in my work, fruitful discussions and careful reading of drafts of this paper. I especially thank Jan Willem Klop for his suggestions, the interpretation of the not-variable-applying condition in terms of orthogonality is due to him.

References

[Bar84] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[Bar90] F. Barbanera. Combining term rewriting and type assignment systems. *Int. J. of Foundations of Computer Science*, 1(3):165–184, 1990.

[BT88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proc. Logic in Computer Science*, pages 82–90. IEEE, 1988.

[BTG89] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *ICALP'89, LNCS 372*, pages 137–150. Springer, 1989.

[CC91] F. Cardone and M. Coppo. Type inference with recursive types: Syntax and semantics. *Information and Computation*, 92:48–80, 1991.

[Cos89] S. Cosmadakis. Computing with recursive types. In *Proc. Logic in Computer Science*, pages 24–38. IEEE, 1989.

[Dou91] D. J. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. In R. V. Book, editor, *Rewriting Techniques and Applications, 4th RTA-91, LNCS 488*, pages 37–48. Springer, 1991.

[JO91] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proc. Logic in Computer Science*, pages 350–361. IEEE, 1991.

[Klo80] J. W. Klop. *Combinatory Reduction Systems*. Mathematical Centre Tracts 127. Mathematisch Centrum (CWI), Amsterdam, 1980.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proc. Logic in Computer Science*, pages 342–349. IEEE, 1991.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–256, 1977.

[Ros73] B. K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. of the ACM*, 20(1):160–187, 1973.

[RV80] J.-C. Raoult and J. Vuillemin. Operational and semantic equivalence between recursive programs. *J. of the ACM*, 27(4):772–796, 1980.

[Toy88] Y. Toyama. Commutativity of term rewriting systems. In K. Fuchi and L. Kott, editors, *Programming of Future Generation Computers II*, pages 393–407. North-Holland, 1988.

Supplement, not contained in the article:
Completing the confluence proof of the example

It remains to show the confluence of our applicative term rewriting system (TRS) alone, without abstractions and β-reduction. The sufficient conditions in Huet: "Confluent reductions: Abstract properties and applications to term rewriting systems" that check only convergence of critical pairs of left-linear TRSs are not applicable here: Huet's Lemma 3.3 is almost applicable (Corollary: Any left-linear parallel closed term rewriting system is confluent), but it demands of the critical pair:
$\mathsf{y\ w} \longleftarrow \mathsf{(pcond\ (0\ x)\ y\ z)\ w} \longrightarrow \mathsf{pcond\ (0\ x)\ (y\ w)\ (z\ w)}$ that there should be a parallel reduction step: $\mathsf{y\ w} \longrightarrow \mathsf{pcond\ (0\ x)\ (y\ w)\ (z\ w)}$. Note that the right term of a critical pair is defined by a reduction *at the root*. The lemma demands a parallel reduction step from the left to the right term, not an arbitrary reduction. But in our example there is only a reduction in the opposite direction. [Toy88, Corollary 3.2] gives a sufficient condition more general than Huet's Lemma 3.3; it is also not applicable here by the same reason.

So we prove termination in addition to the convergence of critical pairs and apply the following well-known lemma:

For every TRS with reduction relation $\longrightarrow$ holds: If $\longrightarrow$ is noetherian (i.e. terminating) and for every critical pair (P, Q) there is M with $P \stackrel{*}{\longrightarrow} M \stackrel{*}{\longleftarrow} Q$, then $\longrightarrow$ is confluent.

This lemma holds also for the restriction of $\longrightarrow$ and of critical pairs to some $\rightarrow$-closed subset of terms. Only condition 3 for $\rightarrow$-closed sets must be changed so that any subterm of M may be replaced by a variable.

To prove termination of our example, we define a mapping φ from terms to $\{2, 3, \ldots\}$ by:

$$
\begin{array}{rcl}
\varphi x &=& \varphi 0 = \varphi 1 = \varphi\,\mathsf{pcond} = 2, \\
\varphi(0M) &=& \varphi(1M) = 2 + \varphi M, \\
\varphi(\mathsf{pcond}\ M) &=& (\varphi M) \cdot 2, \\
\varphi(\mathsf{pcond}\ MN) &=& (\varphi M) \cdot (2 + \varphi N), \\
\varphi(\mathsf{pcond}\ MNP) &=& (\varphi M) \cdot (2 + \varphi N + \varphi P), \\
\text{and } \varphi(MN) &=& (\varphi M) \cdot (\varphi N) \text{ for all other applications.}
\end{array}
$$

By simple computation and some case analysis for rule ($\mathsf{pcond} \rightarrow$) we can show: $M \longrightarrow N \Longrightarrow \varphi M > \varphi N$. Thus there are no infinite reduction chains.

Remark: The same mapping φ can be used to prove termination, and then confluence, of the more complicated TRS of [Cos89]. This system contains, among others, our rule ($\mathsf{pcond} \rightarrow$) and the following:
$\mathsf{pcond}\ (\mathsf{pcond}\ \mathrm{x\ y\ z})\ \mathrm{v\ w} \longrightarrow \mathsf{pcond}\ \mathrm{x}\ (\mathsf{pcond}\ \mathrm{y\ v\ w})\ (\mathsf{pcond}\ \mathrm{z\ v\ w})$.
Only some more equations for new operators must be added there, e.g. $\varphi(\mathsf{fst}\, M) = 2 \cdot (\varphi M)$.

Now 6 typable critical pairs remain to be checked in our example:

$$\begin{array}{lll}
(0\ y) & \longleftarrow \mathsf{pcond}\ (0\ x)\ (0\ y)\ (0\ z) & \longrightarrow 0\ (\mathsf{pcond}\ (0\ x)\ y\ z) \\
(1\ y) & \longleftarrow \mathsf{pcond}\ (0\ x)\ (1\ y)\ (1\ z) & \longrightarrow 1\ (\mathsf{pcond}\ (0\ x)\ y\ z) \\
(0\ z) & \longleftarrow \mathsf{pcond}\ (1\ x)\ (0\ y)\ (0\ z) & \longrightarrow 0\ (\mathsf{pcond}\ (1\ x)\ y\ z) \\
(1\ z) & \longleftarrow \mathsf{pcond}\ (1\ x)\ (1\ y)\ (1\ z) & \longrightarrow 1\ (\mathsf{pcond}\ (1\ x)\ y\ z) \\
y\ w & \longleftarrow \mathsf{pcond}\ (0\ x)\ y\ z\ w & \longrightarrow \mathsf{pcond}\ (0\ x)\ (y\ w)\ (z\ w) \\
z\ w & \longleftarrow \mathsf{pcond}\ (1\ x)\ y\ z\ w & \longrightarrow \mathsf{pcond}\ (1\ x)\ (y\ w)\ (z\ w)
\end{array}$$

Now the question arises: Is it possible to make the termination proof of the example superfluous by some new sort of local test on critical pairs? This will be the subject of forthcoming work.