Notes on rewriting

Robin Cockett
Department of Computer Science
University of Calgary

December 4, 2006

1 Introduction

Rewriting is a fundamental technique both in algebra and in programming languages. These notes are aimed at giving a brief introduction to rewriting theory on algebraic systems.

The main topics covered are:

- Termination using well-founded orderings;
- Local confluence and confluence: Newman's lemma that termination and local confluence implies confluence;
- Equality and confluence: the uniqueness of normal forms;
- The confluence of orthogonal rewrite systems;
- Standardization: normal forms for rewriting sequences;
- A brief discussion of combinatory logic and functional completeness.

2 Rewriting on algebraic systems

An algebraic system is determined by:

- A set of function symbols Ω each of which has an associated arity:

$$\mathsf{arity} : \Omega \longrightarrow \mathbb{N}; f \mapsto \mathsf{arity}(f)$$

- A set of terms, $T_\Omega(X)$ built inductively from a set of variables X as follows:

$$\frac{x \in X}{x \in T_\Omega(X)} \text{ variable}$$

$$\frac{t_1, ..., t_n \in T_\Omega(X) \quad \mathsf{arity}(f) = n}{f(t_1, .., t_n) \in T_\Omega(X)} \text{ function application}$$

A rewriting system on an algebraic system is generated by a set of primitive rewritings

$$R \subseteq T_\Omega(X) \times T_\Omega(X)$$

which generate a relation, which we shall write $t_1 \underset{R}{\longrightarrow} t_2$ on $T_\Omega(X)$ as follows:

$$\frac{(t_1, t_2) \in R \quad \sigma \text{ a substitution}}{t_1[\sigma] \underset{R}{\longrightarrow} t_2[\sigma]} \text{ substitution}$$

$$\frac{s_1, .., \widehat{s_i}, .., s_n \in T_\Omega(X) \quad t \underset{R}{\longrightarrow} t' \quad \mathsf{arity}(f) = n}{f(s_1, ..., t, ..., s_n) \underset{R}{\longrightarrow} f(s_1, ..., t', ..., s_n)} \text{ application}$$

We shall denote by $t \underset{R}{\longrightarrow}^+ t'$ the transitive closure of the above relation and by $t \underset{R}{\longrightarrow}^* t'$ the transitive, reflexive closure of the relation. In the sequel we shall drop the R as it will be understood from the context.

Here are three examples:

1. A monoid is an algebraic system with a binary multiplication, $_ \cdot _$, and a unit, sfe, which is a constant (this means the arity is 0):

$$\mathsf{arity} : \Omega = \{\cdot, \mathsf{e}\} \longrightarrow \mathbb{N}; \quad \begin{array}{lcl} \cdot & \mapsto & 2 \\ & \mapsto & 0 \end{array}$$

   a monoid must satisfy equations which here we orient to create a rewriting system:

$$\begin{aligned} (x \cdot y) \cdot z &\longrightarrow x \cdot (y \cdot z) && (1) \\ \mathsf{e} \cdot x &\longrightarrow x && (2) \\ x \cdot \mathsf{e} &\longrightarrow x && (3) \end{aligned}$$

   Examples of monoids include: the natural numbers under addition (unit is 0), scalars under multiplication (unit is 1), matrices under multiplication (unit is the diagonal matrix).

2. A group is a monoid which adds one more operation called inverse which is a unary operation $(_)^{-1}$ written traditionally as a superscript. We than have:

$$\mathsf{arity} : \Omega = \{\cdot, \mathsf{e}, (_)^{-1}\} \longrightarrow \mathbb{N}; \quad \begin{array}{lcl} \cdot & \mapsto & 2 \\ \mathsf{e} & \mapsto & 0 \\ (_)^{-1} & \mapsto & 1 \end{array}$$

   a group must satisfy equations, which includes those of a monoid which here we orient to create a rewriting system:

$$\begin{aligned} (x \cdot y) \cdot z &\longrightarrow x \cdot (y \cdot z) \\ \mathsf{e} \cdot x &\longrightarrow x \\ x \cdot \mathsf{e} &\longrightarrow x \\ x \cdot x^{-1} &\longrightarrow \mathsf{e} \\ x^{-1} \cdot x &\longrightarrow \mathsf{e} \end{aligned}$$

Neither the natural numbers under addition nor the scalars under multiplication form groups as they lack inverses (the former because of the lack of negative numbers and the latter because 0 does not have an inverse). However, the non-zero scalars (of a field) from a group and the integers under addition form a group. Also the invertible matrices under multiplication form a group.

3. A combinatory algebra consists of a binary operation, called application, written as $_ \bullet _$ and two constants k and s:

$$\mathsf{arity} : \Omega = \{\bullet, \mathsf{k}, \mathsf{s}\} \longrightarrow \mathbb{N}; \quad \begin{array}{lcl} \bullet & \mapsto & 2 \\ \mathsf{k} & \mapsto & 0 \\ \mathsf{s} & \mapsto & 0 \end{array}$$

These must satisfy two equations which we orient:

$$\begin{array}{rcl} (\mathsf{k} \bullet x) \bullet y & \longrightarrow & x \\ ((\mathsf{s} \bullet x) \bullet y) \bullet z & \longrightarrow & (x \bullet z) \bullet (y \bullet z) \end{array}$$

The main example of a combinator algebra which we have is the closed terms of the λ-calculus where application is application and

$$\mathsf{k} = \lambda xy.x \quad \text{and} \quad \mathsf{s} = \lambda xyz.xz(yz).$$

Clearly β-reduction corresponds to the rewritings above ...

3 Termination

A rewriting system is **terminating** if the rewriting relation is **well-founded** in the sense that every non-empty set has a least element (this is a normal form relative to that set). This means equivalently that there are no infinite chain of rewrites

$$t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow t_3 \longrightarrow ...$$

as such would not have a minimal element. So, for example, β-reduction on the λ-calculus is not terminating.

If a relation is well-founded its transitive closure (here $t \xrightarrow{+} t'$) is also well-founded: suppose S is a subset we must find a minimal element in here ... however, we do not change this minimal element if we upclose the set $\Uparrow S = \{t \xrightarrow{+} t' | t' \in S\}$. This set has a minimal element with respect to $\longrightarrow$ which must also therefore be minimal with respect to $\xrightarrow{+}$.

In the case that the number of possible rewrites leaving a term is always finite this means there is a bound on the number of rewrites which a chain leaving that term can have. Having a bound is a strictly stronger property than being well-founded. However, in many rewrite systems finding an explicit bound on the rewrites is quite straightforward.

Lemma 3.1 *A rewriting system $\mathcal{R}$ is terminating if and only if there is a well-founded set $(W, <)$ and a map $\alpha : T_\Omega(X) \longrightarrow W$ such that:*

- $\alpha(r_i) > \alpha(c_i)$ *for each* $ri \longrightarrow c_i \in \mathcal{R}$;

- *If* $\alpha(t) > \alpha(t')$ *then* $\alpha(t[\sigma]) > \alpha(t'[\sigma])$*;*
- *If* $\alpha(t_i) > \alpha(t_i')$ *then* $\alpha(f(t_1, ..., t_i, ..., t_n)) > \alpha(f(t_1, ..., t_i', ..., t_n))$.

PROOF: If the rewriting system is terminating then $t \xrightarrow{=} t'$ is well founded so we take α to be the identity map. Conversely given such an α suppose $\emptyset \neq S \subseteq T_\Omega(X)$ then the set $\{\alpha(s)|s \in S\} \subseteq W$ is non-empty and therefore has a minimal element $\alpha(t_0)$. However if $t_0 \longrightarrow t_0'$ is a reduction step which has $t_0' \in S$ then $\alpha(t_0') < \alpha(t_0)$ contradicting the minimality of $\alpha(t_0)$. Thus, t_0 is a minimal element in S with respect to the reduction order showing this order is well-founded. □

Thus, to demonstrate that a rewrite system terminates it suffices to exhibit a map α satisfying the conditions of this lemma.

The rewrite system for monoids is terminating:

Consider the "cost" function $b : T_\Omega(X) \longrightarrow \mathbb{N}$ given by:

$$\begin{aligned} \alpha(x) &= 1 \\ \alpha(\mathsf{e}) &= 1 \\ \alpha(t \cdot t') &= 2\alpha(t) + \alpha(t') \end{aligned}$$

I claim this gives a bound for the rewriting. To see this it suffices to show that if $t \longrightarrow t'$ then $b(t) < b(t)$ so that each rewriting strictly decreases the cost. As $\mathbb{N}$ is well-founded this ensures the rewriting is well founded.

Note that (a) each terms has cost at least 1 (b) decreasing the cost of a subterm will result in at least that amount of cost decrease for the whole term. This means that it suffices to check that the rewrites applied at the root of the term always strictly decrease the cost. However, this is a simple calculation:

$$\begin{aligned} \alpha((t_1 \cdot t_2) \cdot t_3) &= 4\alpha(t_1) + 2\alpha(t_2) + \alpha(t_3) > 2\alpha(t_1) + 2\alpha(t_2) + \alpha(t_3) = \alpha(t_1 \cdot (t_2 \cdot t_3) \\ \alpha(\mathsf{e} \cdot t) &= 2 + \alpha(t) > \alpha(t) \\ \alpha(t \cdot \mathsf{e}) &= 2\alpha(t) + 1 > \alpha(t). \end{aligned}$$

The grandfather of all well-founded sets is the natural numbers with the usual (strict) ordering (this is also a total order) and there are a number of important ways of constructing well-founded posets:

Lexicographical orderings on strings: If P is well-founded then P^* with the lexicographical ordering. The lexicographical ordering is given by [] is minimal and $x : xs < y : ys$ if either $x < y$ or if $x = y$ and $xs < ys$.

This strict inequality is well-founded as given any $X \subset P^*$ if the empty list is in there is automatically a minimal element otherwise choose a minimal element $x_0 \in P$ in the first entry and consider the tails $\mathsf{Tail}_{x_0}(X) = \{xs|x_0 : xs \in X\}$, if there is a minimal element in this set then there is a minimal element in X: in fact, if we continue this process this amounts to asking that one of the tails sets contains the empty list. Suppose for contradiction that this never happens then we would have an element $p_0 \in P^*$ which is in all the sets

$$X \supseteq x_0 : \mathsf{Tail}_{x_0}(X) \supseteq x_0 : x_1 : \mathsf{Tail}_{x_0}(\mathsf{Tail}_{x_0}(X)) \supseteq x_0 : x_1 : x_2 : \mathsf{Tail}_{x_2}(\mathsf{Tail}_{x_1}(\mathsf{Tail}_{x_0}(X))) \supseteq ...$$

but p_0 has a finite length so one of these sets would contain the empty list.

Lexicographical ordering of Rose trees: We consider trees with entries at the nodes and a finite list of subtrees (that is `data Tree a = (;) a [Tree a]`) these are called Rose trees. Rose trees of a well-founded poset form a well-founded set. We set $x; xs < y; ys$ if $x < y$ or if $x = y$ and $xs < ys$ (where we use the lexicographical ordering of the subtrees). The argument that this strict ordering is well-founded is almost identical to the argument we have just done: in any set of rose trees there are trees with a minimal element element at the root we may than consider the set of lists of arguments $\mathsf{Tails}_{x_0}(X)$ if this contains a minimal element we are done. At any rate either this contains the empty list (so we are done) or we may choose a lexicographically least list of first elements. This allows us to grow a prefix of a Rose tree agreeing with elements in X which are minimal so far. The same argument as before shows that this growing process must close off with empty lists of arguments as a given Rose tree which is in all these sets is finite.

Bags: A bag is an unordered list: this means repetitions are allowed: $\{\!\{x, y, x\}\!\} = \{\!\{y, x, x\}\!\} \neq \{\!\{x, y\}\!\}$ but the order in which the elements occur does not matter. The set $\mathsf{Bag}(P)$ of bags of elements of a well-founded set form a well-founded set. One bag is less than another in case

- The empty bag $\{\!\{\}\!\}$ is minimal.
- $\{\!\{x\}\!\} \sqcup b_1 < \{\!\{x\}\!\} \sqcup b_2$ if $b_1 < b_2$;
- $\{\!\{x_1, .., x_n\}\!\} \sqcup b_1 < \{\!\{y\}\!\} \sqcup b_2$ whenever each $x_i < y$.

Thus we may determine whether one bag is (strictly) less than the other by first removing the elements in common and then for each element in the bigger bag removing all elements in the smaller bag which are strictly less than it. The smaller bag will be emptied by this process.

Intuitively to construct an element strictly smaller than a given b we are allowed to pick an element from the bag and either remove it or replace it with a bag of elements each of which is individually strictly smaller than the element removed. It is not so obvious that this process must always end as you may replace the element with a very large bag of smaller elements!

To prove this is well-founded is not so easy (Dershowitz and Manna). Here is a sketch: suppose there is an infinite descending chain of bags $b_0 < b_1 < b_2 < ...$ then we may build a forest with roots $x \in b_0$ the children of x are the bag $\{x_1, .., x_n\}$ where $b_1 < b_{i+1}$ (for some i) introduces this strict replacement of x (this may never happen in which case x is a leaf). Every step in the sequence must do one (or more) replacements of this nature. However, this tree is finitely branching and has all its paths of finite length so it is finite. But this means the sequence itself must be finite.

Unordered trees: An unordered tree is a rose tree in which we do not care about the order of the children. (that is morally `data UTree a = (.)  a (Bag (Utree a))`). if P is a well-founded set then $\mathsf{UTree}(P)$ is also well-founded. The order is given inductively using the bag ordering: $x.\{\!\{t_1, .., t_n\}\!\} \leq y.\{\!\{s_1, ..., s_m\}\!\}$ if $x < y$ or when $x = y$ when $\{\!\{t_1, .., t_n\}\!\} < \{\!\{s_1, ..., s_m\}\!\}$.

Now, in fact, an unordered tree may equally be represented by its bag of paths. The ordering I have just described is just the bag ordering on the lexicographical ordering on the paths! Therefore it is certainly well-founded.

Recursive path orderings: There is a more sophisticated family of well-founded ordering on unordered trees which are called *recursive path orderings* they may be given (following Klop) by the transitive closure of a rewrite system between unordered trees (where one adds a unary operation $(_)^*$) restricted to the unordered trees (so you must eventually remove the added symbol!).

$$\begin{array}{rcll} n.b & \longrightarrow & n^*.b & \\ n^*.b & \longrightarrow & m.\{\!\{n^*.b, ..., n^*.b\}\!\} & m < n \\ n^*.\{\!\{m.b\}\!\} \sqcup b' & \longrightarrow & n.\{\!\{m^*.b, ..., m^*b\}\!\} \sqcup b' & \text{(where no copies is permitted)} \\ n^*.\{\!\{m.b\}\!\} \sqcup b' & \longrightarrow & m.b & \end{array}$$

See Klop for the proof.

Exercise: can you show that the rewriting system determined on terms by $x \cdot (x + y) \longrightarrow (x \cdot x) + (x \cdot y)$ and $(x + y) + z \longrightarrow x + (y + z)$ is terminating?

An illustration of recursive path ordering (Klop) is for the rewrite system

$$\begin{array}{rcl} \neg\neg y & \longrightarrow y & \\ \neg(x \vee y) & \longrightarrow & (\neg x) \wedge (\neg y) \\ \neg(x \wedge y) & \longrightarrow & (\neg x) \vee (\neg y) \\ x \wedge (y \vee z) & \longrightarrow & (x \wedge y) \vee (x \wedge z) \\ (y \vee z) \wedge x & \longrightarrow & (y \wedge x) \vee (z \wedge x) \end{array}$$

We map terms to unordered trees on the natural numbers:

$$V(x \vee y) = 1.\{\!\{V(x), V(y)\}\!\} \quad V(x \wedge y) = 2.\{\!\{V(x), V(y)\}\!\} \quad V(\neg x) = 3.\{\!\{V(x)\}\!\}$$

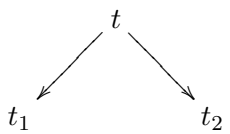
then check that each rule is a reduction here are some of the verifications (exercise complete the set):

$$\begin{array}{rcl} \neg\neg x & = & 3.\{\!\{3.\{\!\{V(x)\}\!\}\}\!\} \longrightarrow 3^*.\{\!\{3.\{\!\{V(x)\}\!\}\}\!\} \longrightarrow 3.\{\!\{V(x)\}\!\} \longrightarrow 3^*.\{\!\{V(x)\}\!\} \longrightarrow V(x) \\ \neg(x \vee y) & = & 3.\{\!\{1.\{\!\{V(x), V(y)\}\!\}\}\!\} \longrightarrow 3^*.\{\!\{1.\{\!\{V(x), V(y)\}\!\}\}\!\} \\ & \longrightarrow & 1.\{\!\{3^*.\{\!\{1.\{\!\{V(x), V(y)\}\!\}\}\!\}, 3^*.\{\!\{1.\{\!\{V(x), V(y)\}\!\}\}\!\}\}\!\} \\ & \longrightarrow & 2.\{\!\{3.\{\!\{1^*.\{\!\{V(x), V(y)\}\!\}\}\!\}, 3.\{\!\{1^*.\{\!\{V(x), V(y)\}\!\}\}\!\}\}\!\} \\ & \longrightarrow & 2.\{\!\{3.\{\!\{V(x)\}\!\}, 3.\{\!\{V(y)\}\!\}\}\!\} = V((\neg x) \wedge (\neg y)) \end{array}$$

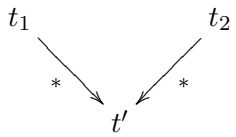
It is worth noting that recursive path ordering cannot, in general, handle associative laws. One way to recover this ability is to add subscripts to the operation indicating its left depth (for example) ...

4 Confluence

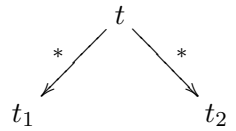
We shall say that a rewriting system is **locally confluent** if every one-step divergence



has a (possibly) multi-step convergence:

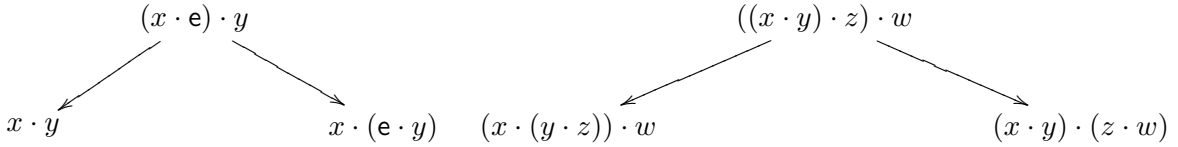


We shall say that a rewriting is **confluent** if every multi-step divergence

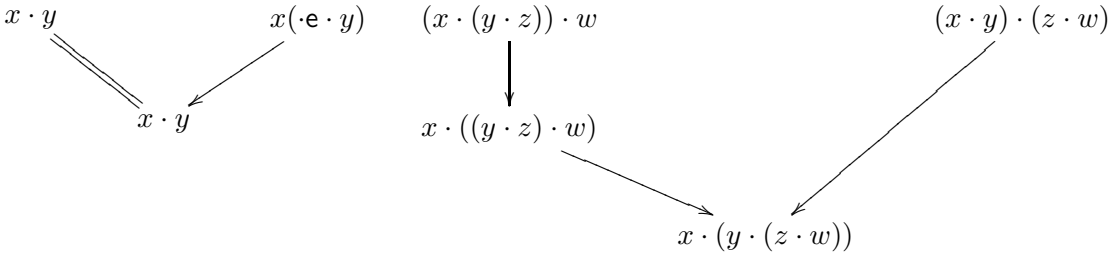


has a convergence as above. Some examples of divergences are:

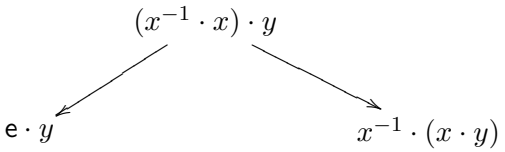
1. Two one-step divergences from the example of monoids are:



   For each there are, in this case, corresponding convergences. Note that the first uses the reflexivity non-trivially while the second requires more than on rewriting step on one of the legs:



2. A one step divergence from the example of groups is:



   Notice that this divergence has no corresponding convergence with respect to the rewrite system given so that this rewrite system is not confluent.

With respect to a rewriting relation a term t is said to be in **normal form** if it is minimal in the sense that there is no rewriting $t \longrightarrow t'$.

When rewrite systems become more sophisticated then the fundamental use termination in the sense of the rewrites being well-foundedness becomes more essential. To show termination of simple rewrite systems, often, a cost function, as above, will do the trick. Assuming that the one-step rewriting is well-founded forces us to do well-founded inductions which use the following principal:

Principle of well-founded induction: Given a well-founded relation (here the one-step rewriting or its transitive closure) we have the following inference for any property:

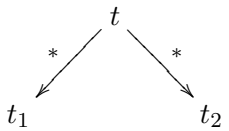
$$\frac{\forall t.(\forall t'.t > t' \Rightarrow P(t')) \Rightarrow P(t)}{\forall t.P(t)}\ \text{wf-ind}$$

Notice that this means that the property must be true, in particular, for all t which are minimal (i.e. are in normal form) as these have no elements below them.

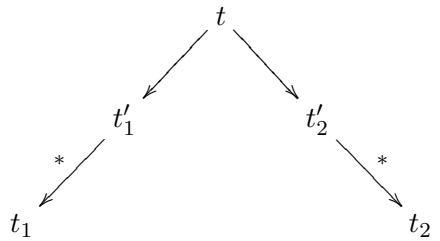
Lemma 4.1 (Newman) *Any terminating rewriting system which is locally confluent is confluent.*

Proof: The property P we wish to prove is "every divergence leaving t has a convergence. This is therefore certainly true of any elements in normal form. Suppose t is such that whenever t' has $t \xrightarrow{+} t'$ then t' has this property then we argue as follows to show that t has the property:

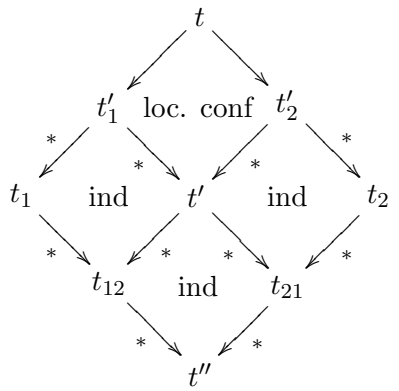
Consider a divergence



if either of the arms is trivial (i.e involves no steps) then there is an obvious convergence. So we may assume that each arm has at least one step. So the divergence is now of the form:



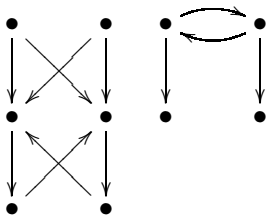
However, everything strictly below t satisfies the induction hypothesis so that:



□

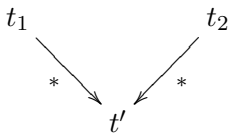
Notice that if a system is confluent then each t has at most one normal form associated with it (there may be none: remember the λ-calculus). This is because suppose that t has two distinct normal forms s_1 and s_2 with $t \stackrel{*}{\longrightarrow} s_1$ and $t \stackrel{*}{\longrightarrow} s_2$ then this constitutes a divergence. However the convergence forces $s_1 = s_2$. So in a confluent system normal forms are unique.

Here is a rewriting system (between constants in an algebraic theory), in which local confluence holds yet confluence does not hold, in which there are objects with no normal form and with two normal forms (the example follows Huet):



Given a rewriting system we always regard treat the rewrites as equalities. This gives the following observation which explains the preoccupation with confluence:

Lemma 4.2 *In a confluent rewriting system $t_1 = t_2$ if and only if there is a convergence*

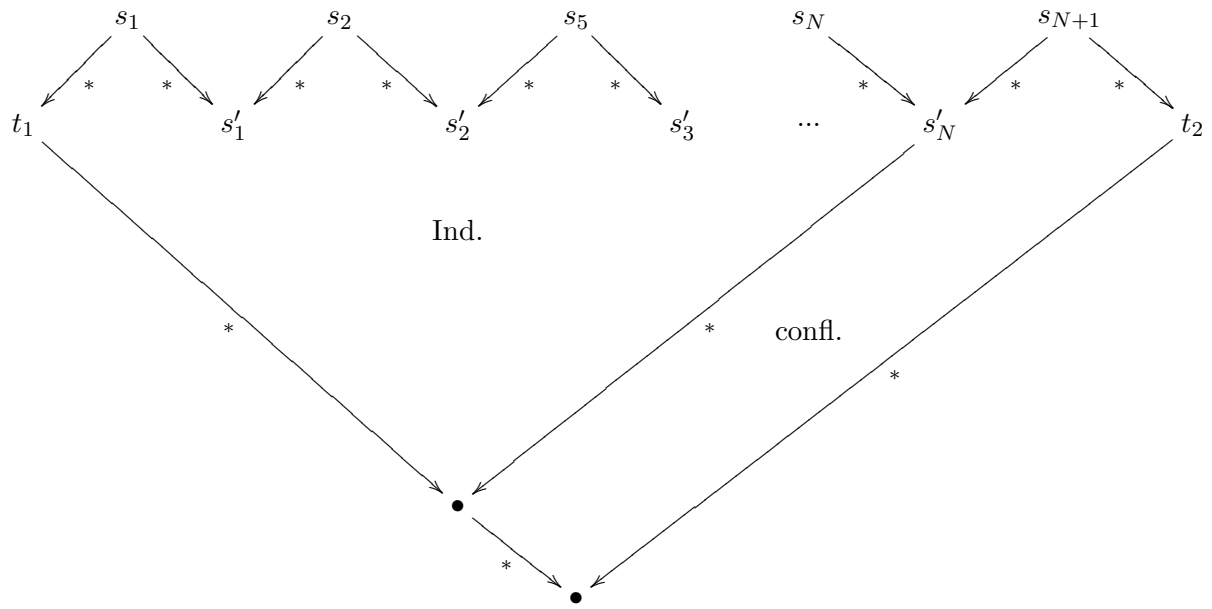


PROOF: t_1 equals t_2 if and only if there is a zigzag of rewritings:

$$t_1 \stackrel{*}{\longleftarrow} s_1 \stackrel{*}{\longrightarrow} s_1' \stackrel{*}{\longleftarrow} s_2 \stackrel{*}{\longrightarrow} s_2' \stackrel{*}{\longleftarrow} ...s_n \stackrel{*}{\longrightarrow} t_2$$

We argue by the number of zigs (in this case n). If n is 0 the result is immediate. It is nice, though not necessary, to observe that when $n = 1$ then it is given by the confluence. We suppose the result

true for N and consider a zigzag with $N+1$ zigs.



Using the induction hypothesis and confluence shows that the result holds for N. □

Corollary 4.3 *In a confluent terminating rewriting system*

(i) *Every term has a normal form;*

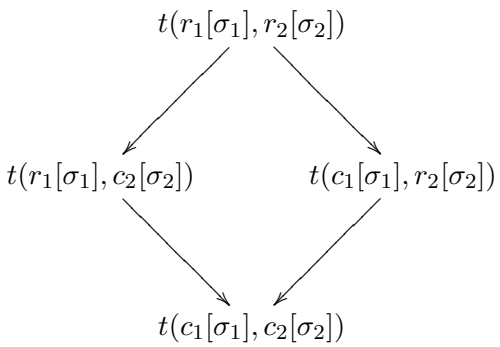
(ii) *Two terms are equal if and only if they have the same normal form.*

5 Critical pairs

In an algebraic system there are (usually) infinitely many terms so that it is not practical to check every divergence for a convergence. It is therefore important to understand which (one step) divergences one actually has to check. To understand this it is useful to understand what a **redex** is: it is the part of the term that a rewriting rule removes in order to replace it with the **contractum**. A redex indicates where in the term a primitive rewriting could be applied and the part of the subterm standing at that place which will be affected by the rewriting.

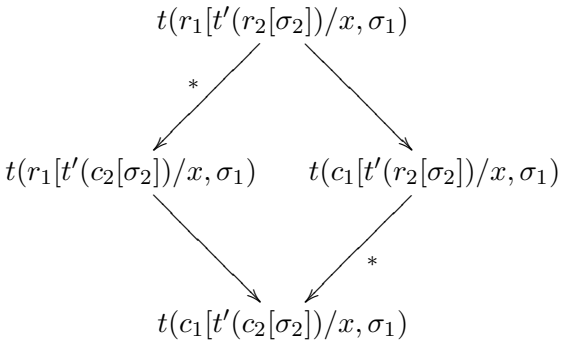
We say that two rewritings $t \longrightarrow t_1$ and $t \longrightarrow t_2$ are **independent** in case the redexes of the primitive rewritings they embody are disjoint. This can happen in two different ways: firstly the redexes can be in parallel parts of the term (i.e. they are below different arguments of some function symbol). We may write this situation as $t(r_1[\sigma_1], r_2[\sigma_2])$ where $r_1 \longrightarrow c_1$ and $r_2 \longrightarrow c_2$ are the two

rewritings. In this situation it is clear that:



so that local confluence is assured.

The other possibility is that one redex is below the other which we represent as $t(r_1[t'(r_2[\sigma_2])/x, \sigma_1)$. There is immediately a little subtly concerning this situation which we should be clear about: in the redex r_1 the variable x may occur more an once. Thus r_2 may occur in multiple (parallel) places and it is only one of these redexes which is the actual second redex we had in mind when we started. However, we do have the following:



and from this we can obtain our local confluence by replacing the downward arrow by a sequence of rewrites the first of which is the chosen rewrite.

We have now shown that:

Lemma 5.1 *In any rewrite system over an algebraic system, rewrites with independent redexes are locally confluent.*

The importance of this is that in order to establish local confluence it suffices to check the cases when the redexes overlap. However, we can go much further: clearly all the rewriting takes place on the term below where the lower redex starts so we need only consider the subterm at which the lower redex starts. Finally, notice that the terms which are just the overlapping redexes themselves will also exhibit the divergence (in fact, in a minimal way). Furthermore, when this divergence has a convergence any substitution of it will have a convergence. Thus, by solving such a divergence, which we certainly must do anyway, we will have solved all the cases where this pattern of overlapping redexes occur. Thus, it suffices to check the divergences arising from these overlapping redexes for convergence. These minimal divergences are called **critical divergences**.

When the rewriting system is terminating we may rewrite the two feet of a critical divergence into normal forms a pair of such terms is called a **critical pair**

To form critical pairs involves identifying a subterm in a redex (which can be the whole term) at which the overlap will happen. Thus given a redex r_1 one splits it into $r_1 = r_1'(s)$, where s is not a variable, and next one finds a most general substitution, that is a the most general unifier, σ, such that $r_2[\sigma] = s[\sigma]$, where r_2 and s are assumed to have no variables in common. The divergence:

$$\begin{array}{ccc} & r_1[\sigma] = r_1'(s)[\sigma] & \\ \swarrow & & \searrow \\ c_1[\sigma] & & r_1'(c_2)[\sigma] \end{array}$$

is then a critical pair. Notice that this whole construction is more delicate as r_1 can have repeated variables, as in the rewrite $x \cdot x^{-1} \longrightarrow sfe$ above for example, so that simply overlapping r_1 and r_2 in the obvious manner may destroy the redex for r_2 as it requires that two subterms are equal. Using unification maintains this and delivers the minimal way to do this. Thus, the critical divergence from $(x \cdot y) \cdot z$ and $x \cdot x^{-1} \longrightarrow sfe$ is

$$x \cdot (y \cdot (x \cdot y)^{-1}) \longleftarrow (x \cot y) \cdot (x \cdot y)^{-1} \longrightarrow \mathsf{e}.$$

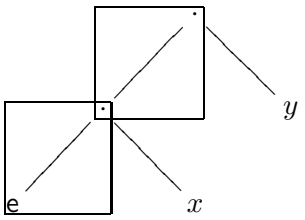
Proposition 5.2 *A rewrite system over an algebraic system is locally confluent if and only if all critical divergences can be converged.*

This reduces testing a (finitely presented) system for local confluence to a finite number of tests. Thus it is an easily decidable question.

Consider the rewriting system for monoids:

At this stage we know the rewriting system is terminating so it remains to look at the *critical pairs* that is minimal terms in which there are overlapping redexes. It is useful here to visualize the terms as trees so that one can see what is going on. There are five critical pairs:

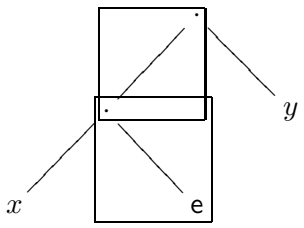
1. Rules (2) and (1) have a critical pair arising from the following overlapping redexes:



This critical pair is resolved as follows:

$$\begin{array}{ccc} (\mathsf{e} \cdot x) \cdot y & \xrightarrow{(1)} & \mathsf{e} \cdot (x \cdot y) \\ {\scriptstyle (2)}\downarrow & & \downarrow{\scriptstyle (2)} \\ x \cdot y & = & x \cdot y \end{array}$$

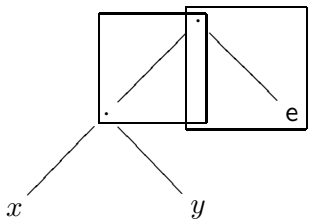
2. Rules (3) and (1) have a first critical pair arising from the following overlapping redexes:



This critical pair is resolved as follows:

$$\begin{array}{ccc} (x \cdot \mathsf{e}) \cdot y & \xrightarrow{(1)} & x \cdot (\mathsf{e} \cdot y) \\ {\scriptstyle (3)}\downarrow & & \downarrow{\scriptstyle (2)} \\ x \cdot y & = & x \cdot y \end{array}$$

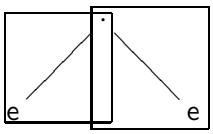
3. Rules (1) and (3) have a critical pair arising from the following overlapping redexes:



This critical pair is resolved as follows:

$$\begin{array}{ccc} (x \cdot y) \cdot \mathsf{e} & \xrightarrow{(1)} & x \cdot (y \cdot \mathsf{e}) \\ {\scriptstyle (3)}\downarrow & & \downarrow{\scriptstyle (3)} \\ x \cdot y & = & x \cdot y \end{array}$$

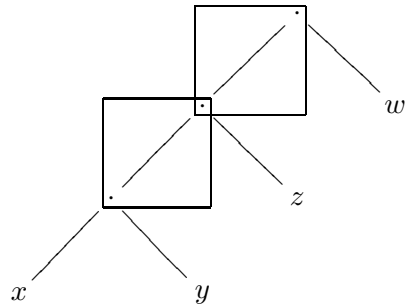
4. Rules (3) and (2) have a critical pair arising from the following overlapping redexes:



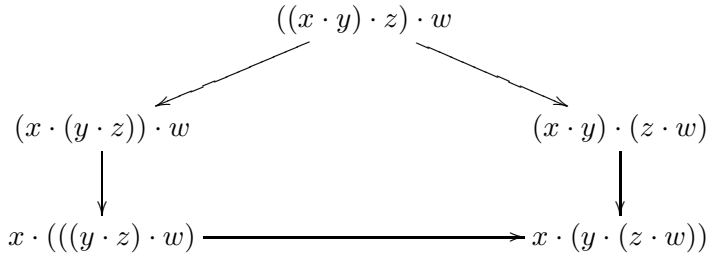
This critical pair is (trivially) resolved as follows:

$$\begin{array}{ccc} \mathsf{e} \cdot \mathsf{e} & \xrightarrow{(3)} & \mathsf{e} \\ {\scriptstyle (2)}\downarrow & & \| \\ \mathsf{e} & = & \mathsf{e} \end{array}$$

5. Rules (1) and (1) have a critical pair arising from the following overlapping redexes:



This critical pair is resolved as follows (this is actually an instance of something very famous sometimes called "the MacLane pentagon"):



The example of the divergence mentioned earlier for the rewriting system for groups given above shows that the rewriting system on groups, as given, is not confluent.

A natural further step is to try to complete a rewriting system to make it confluent. This is the Knuth-Bendix completion procedure: the basic idea is as follows: suppose one starts with a well-founded rewrite system on the terms which agrees with given rewrites and one discovers that a critical divergence cannot be resolved then one adds a rewrite to the system across the pair oriented in the direction of the well-order. This results in an expanded system of rewrites, next one removes any rewrites which are now implied. One then keeps going like this until there are no unresolved critical pairs.

Of course there is no guarantee that this procedure will terminate. However, it gives a very effective way of searching for a confluent rewrite system which often does terminate.

Consider the example of groups. We know the rewriting system as it stands is not confluent as certain critical pairs cannot be resolved:

$$\begin{aligned}
\mathsf{e} \cdot y \leftarrow \quad & (x^{-1} \cdot x) \cdot y \quad \longrightarrow x^{-1} \cdot (x \cdot y) \\
\mathsf{e} \cdot y \leftarrow \quad & (x \cdot x^{-1}) \cdot y \quad \longrightarrow x \cdot (x^{-1} \cdot y) \\
\mathsf{e} \leftarrow \quad & \mathsf{e} \cdot \mathsf{e}^{-1} \quad \longrightarrow \mathsf{e}^{-1} \\
\mathsf{e}^{-1} \leftarrow \quad & \mathsf{e}^{-1} \cdot \mathsf{e} \quad \longrightarrow \mathsf{e}
\end{aligned}$$

These cause us to add to the system the oriented equations:

$$\begin{aligned}
x^{-1} \cdot (x \cdot y) \quad &\longrightarrow \quad y \\
x \cdot (x^{-1} \cdot y) \quad &\longrightarrow \quad y \\
e^{-1} \quad &\longrightarrow \quad e
\end{aligned}$$

(where notice we have reduced each term in the critical pairs to a normal form). Adding these equations introduce more critical pairs in particular:

$$\begin{array}{rll} x \cdot e \leftarrow & x \cdot (x^{-1} \cdot (x^{-1})^{-1}) & \rightarrow (x^{-1})^{-1} \\ x \cdot (y \cdot (x \cdot y)^{-1} \leftarrow & (x \cdot y) \cdot (x \cdot y)^{-1} & \rightarrow \mathsf{e} \end{array}$$

which causes us to add (among other things)

$$\begin{array}{rcl} (x^{-1})^{-1} & \rightarrow & x \\ x \cdot (y \cdot (x \cdot y)^{-1} & \rightarrow & e \end{array}$$

This gives

$$y \cdot (x \cdot y)^{-1} \leftarrow \cdot x^{-1} \cdot (x \cdot (y \cdot (x \cdot y)^{-1}) \rightarrow x^{-1} \cdot \mathsf{e}$$

which causes the addition of the rewrite

$$y \cdot (x \cdot y)^{-1} \rightarrow x^{-1}$$

this rewrite gives rise to the critical pair

$$(x \cdot y)^{-1} \leftarrow y^{-1} \cdot (y \cdot (x \cdot y)^{-1}) \rightarrow y^{-1} \cdot x^{-1}.$$

which causes the rewrite

$$(x \cdot y)^{-1} \rightarrow y^{-1} \cdot x^{-1}$$

to be added. Finally this gives a confluent set of rewritings for groups:

$$\begin{array}{rcl} (x \cdot y) \cdot z & \rightarrow & x \cdot (y \cdot z) \\ \mathsf{e} \cdot x & \rightarrow & x \\ x \cdot \mathsf{e} & \rightarrow & x \\ x \cdot x^{-1} & \rightarrow & \mathsf{e} \\ x^{-1} \cdot x & \rightarrow & \mathsf{e} \\ \mathsf{e}^{-1} & \rightarrow & \mathsf{e} \\ x \cdot (x^{-1} \cdot y) & \rightarrow & y \\ x^{-1} \cdot (x \cdot y) & \rightarrow & y \\ (x^{-1})^{-1} & \rightarrow & x \\ (x \cdot y)^{-1} & \rightarrow & y^{-1} \cdot x^{-1} \end{array}$$

Exercise: show that this is locally confluent and terminating.

6 Orthogonal systems

A very basic result in rewriting theory is that all orthogonal rewriting systems are confluent. A rewrite rule is **orthogonal** in case it is left linear and non-overlapping. It is **left-linear** in case the lefthand side does not contain any repeated variables (recall the problems these gave above) and contains all the variable of the righthand side. The rewriting system is **non-overlapping** in case the redexes never overlap unless they are the same: that is there are no critical pairs. An example of such a rewriting system is the system for combinatory algebra above.

Theorem 6.1 *Every orthogonal system is confluent.*

It is clear that such a system must be locally confluent, so if the system is terminating, it already implies that the system is confluent. However, proving termination can be non-trivial so that even when the system is terminating this is a useful observation. The force of this result, however, is for non-terminating systems:

Corollary 6.2 *The rewriting system on combinatory algebra is confluent.*

The remainder of this section is dedicated to proving this result in some detail. We shall use a labelling argument similar to the the one used to establish the confluence of the λ calculus.

Let $\mathcal{R} = \{r_i \longrightarrow c_i | i \in I\}$ be a left-linear orthogonal rewriting system on a set of terms $T_\Omega(X)$. Then we may augment the system with a set of labelled operations corresponding to the redexes of the rewrite rules

$$\Omega(\mathcal{R}) = \Omega \cup \{\lambda^*(x_1, ..., x_n).r_i | r_i \longrightarrow c_i \in \mathcal{R}, FV(r_i) = \{x_1, ..., x_n\}\}$$

to get a set of terms $T_{\Omega(\mathcal{R})}(X)$. In addition we may add a set of labelled rewrites corresponding to the original rewrites but starting at the labelled operations

$$\mathcal{R}^* = \mathcal{R} \cup \{(\lambda^*(x_1, ..., x_n).r_i)(x_1, ..., x_n) \longrightarrow c_i | r_i \longrightarrow c_i \in \mathcal{R}\}.$$

Thus we regard $\lambda^*(x_1, ..., x_n).r_i$ as a new operation symbol which has arity n for the system which also has associated rewrite rules as above.

There is an obvious mapping $\varepsilon : T_{\Omega(*\mathcal{R})}(X) \longrightarrow T_\Omega(X)$ which simply removes the labelling. This is defined by:

$$\begin{aligned} \varepsilon(x) &= x \qquad x \in X \\ \varepsilon(f(t_1, ..., t_n)) &= f(\varphi(t_1), ..., \varphi(t_n)) \\ \varepsilon((\lambda^*(x_1, ..., x_n).r_i)(t_1, ..., t_n)) &= r_i[\varepsilon(t_1)/x_1, ..., \varepsilon(t_n)/x_n] \end{aligned}$$

Given a term t in this labelled system we may also define a function $\varphi : T_{\Omega(\mathcal{R})}(X) \longrightarrow T_\Omega(X)$ which performs all the labelled rewrites as follows:

$$\begin{aligned} \varphi(x) &= x \qquad x \in X \\ \varphi(f(t_1, ..., t_n)) &= f(\varphi(t_1), ..., \varphi(t_n)) \\ \varphi((\lambda^*(x_1, ..., x_n).r_i)(t_1, ..., t_n)) &= c_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n] \end{aligned}$$

Notice this is a "by-value" evaluation strategy.

An important observation concerning this latter function is:

Lemma 6.3 $\varphi(t[t'/x]) = \varphi(t)[\varphi(t')/x]$.

PROOF: We do a structural induction on t. There are three cases to consider:

(a) t is a variable: in this case the result is immediate.

(b) $t = f(t_1, ..., t_n)$ then we have:

$$\begin{aligned}
\varphi(f(t_1, ..., t_n))[\varphi(t')/x] &= f(\varphi(t_1), ..., \varphi(t_n))[\varphi(t')/x] \\
&= f(\varphi(t_1)[\varphi(t')/x], ..., \varphi(t_n)[\varphi(t')/x]) \\
&= f(\varphi(t_1[t'/x]), ..., \varphi(t_n[t'/x])) \\
&= \varphi(f(t_1[t'/x], .., t_n[t'/x])) \\
&= \varphi(f(t_1, .., t_n)[t'/x])
\end{aligned}$$

(c) $t = (\lambda^*(x_1, ..., x_n).r_1)(t_1, .., t_n)$ then we have:

$$\begin{aligned}
\varphi((\lambda^*(x_1, ..., x_n).r_1)(t_1, ..., t_n))[\varphi(t')/x] &= c_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n][\varphi(t')/x] \\
&= c_i[\varphi(t_1)[\varphi(t')/x]/x_1, ..., \varphi(t_n)[\varphi(t')/x]/x_n] \\
&= c_i[\varphi(t_1[t'/x])/x_1, ..., \varphi(t_n[t'/x])/x_n] \\
&= c_i[\varphi(t_1[t'/x])/x_1, ..., \varphi(t_n[t'/x])/x_n] \\
&= \varphi(c_i[t_1[t'/x]/x_1, ..., t_n[t'/x]/x_n]) \\
&= \varphi(c_i[t_1/x_1, ..., t_n/x_n][t'/x]).
\end{aligned}$$

□

This has an important consequence:

Lemma 6.4 *The following are valid rewriting diagrams:*

(i)

$$\begin{array}{ccc}
r_i[t_1/x_1, ..., t_n/x_n] & \xrightarrow[*]{\varphi} & \varphi(r_i[t_1/x_1, ..., t_n/x_n]) \\
{\scriptstyle r_i}\downarrow & & \downarrow{\scriptstyle r_i} \\
c_i[t_1/x_1, ..., t_n/x_n] & \xrightarrow[\varphi]{*} & \varphi(c_i[t_1/x_1, ..., t_n/x_n])
\end{array}$$

(ii)

$$\begin{array}{ccc}
t[r_i[t_1/x_1, ..., t_n/x_n]/y] & \xrightarrow[*]{\varphi} & \varphi(t[r_i[t_1/x_1, ..., t_n/x_n]/y]) \\
{\scriptstyle t[r_i/y]}\downarrow * & & * \downarrow{\scriptstyle \varphi(t)[r_i/y]} \\
t[c_i[t_1/x_1, ..., t_n/x_n]/y] & \xrightarrow[\varphi]{*} & \varphi(t[c_i[t_1/x_1, ..., t_n/x_n]/y])
\end{array}$$

(iii)

$$\begin{array}{ccc}
t[(\lambda\tilde{x}.r_i)(t_1, ..., t_n)/y] & & \\
{\scriptstyle t[(\lambda\tilde{x}.r_i)/y]}\downarrow * & \searrow^{\varphi}_{*} & \\
t[c_i[t_1/x_1, ..., t_n/x_n]/y] & \xrightarrow[\varphi]{*} & \varphi(t[(\lambda\tilde{x}.r_i)(t_1, ..., t_n)/y])
\end{array}$$

PROOF:

(i) This follows as $\varphi(r_i[t_1/x_1, ..., t_n/x_n]) = r_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n]$ and $\varphi(c_i[t_1/x_1, ..., t_n/x_n]) = c_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n]$ as there is no labelled material in either r_i or c_i.

(ii) This time we make use of the above lemma as:

$$\varphi(t[r_i[t_1/x_1, ..., t_n/x_n]/y]) = \varphi(t)[r_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n]/y]$$

and

$$\varphi(t[c_i[t_1/x_1, ..., t_n/x_n]/y]) = \varphi(t)[c_i[\varphi(t_1)/x_1, ..., \varphi(t_n)/x_n]/y].$$

(iii) We need to show

$$\varphi(t[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y]) = \varphi(t[c_i(t_1, ..., t_n)/y]).$$

The proof is by a structural induction on t. There are three cases:

(a) If t is a variable then either it is y and one can directly use the definition of φ or it is distinct and the result is immediate.

(b) $t = f(t'_1, ..., t'_n)$ where the result holds for t'_1, ... t'_n then we have:

$$\begin{aligned}
&\varphi(f(t'_1, ..., t'_n)[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y]) \\
&\quad = \varphi(f(t'_1[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y], ..., t'_n[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y])) \\
&\quad = f(\varphi(t'_1[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y]), ..., \varphi(t_n[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y])) \\
&\quad = f(\varphi(t'_1)[c_i[x_1/\varphi(t_1), ..., x_n/\varphi(t_n)]/y]), ..., \varphi(t'_n)[c_i[\varphi(t_1)/x_1, ..., \varphi(t_n))/y])) \\
&\quad = f(\varphi(t'_1)[c_i[x_1/\varphi(t_1), ..., x_n/\varphi(t_n)]/y]), ..., \varphi(t'_n)[c_i[\varphi(t_1)/x_1, ..., \varphi(t_n))/y])) \\
&\quad = \varphi(f(t'_1, ..., t'_n))[c_i[x_1/\varphi(t_1), ..., x_n/\varphi(t_n)]/y]
\end{aligned}$$

(c) Lastly for $t = (\lambda^*\tilde{y}.r_j)(t'_1, ..., t'_n)$ we have:

$$\begin{aligned}
&\varphi((\lambda * \tilde{y}.r_j)(t'_1, ..., t'_m)[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y]) \\
&\quad = \varphi((\lambda * \tilde{y}.r_j)(t'_1[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y], ..., t'_m[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y])) \\
&\quad = c_j[\varphi(t'_1[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y])/y_1, ..., \varphi(t'_n[(\lambda^*\tilde{x}.r_i)(t_1, ..., t_n)/y])/y_m] \\
&\quad = c_j[\varphi(t'_1)[c_i[\varphi(t_1)/x_1, ...\varphi(t_n)/x_n]/y])/y_1, ..., \varphi(t'_n)[c_i[\varphi(t_1)/x_1, ...\varphi(t_n)/x_n]/y]/y_m] \\
&\quad = \varphi(c_j[t'_1/y_1, ..., t_m/y_m])[c_i[\varphi(t_1)/x_1, ...\varphi(t_n)/x_n]/y].
\end{aligned}$$

□

Notice the above lemmas do not use the assumption that the rewrites are left linear or non-overlapping.

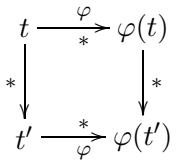
Now suppose that we wish to determine whether a divergence of the form:

$$\begin{array}{ccc} t & \longrightarrow & t'' \\ {\scriptstyle *}\downarrow & & \\ t'' & & \end{array}$$

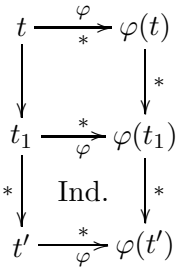
Then we may label the one step rewriting so that it may be viewed as $t \xrightarrow{\varphi} \varphi(t) = t''$ Note that in doing this labelling we may have to turn some of the rewritings on the multi-step leg into labelled

rewritings. Here the importance of being non-overlapping comes into play as we must be assured that by labelling a redex we do not block a reduction but merely are forced to label it. Furthermore, introducing a label in this manner can stop a rule which is not left-linear from being applicable. So if the rules are not left-linear there is no guarantee that after labelling we can reconstruct the rewriting. Thus, at this stage we use both the assumptions of orthogonality.

Once the labelling has been done we may argue using the induction hypothesis that when the multi-step rewriting (down) has length N or less then there is a convergence of the form:

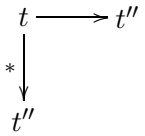


Considering an $N+1$ step rewriting we may split it as

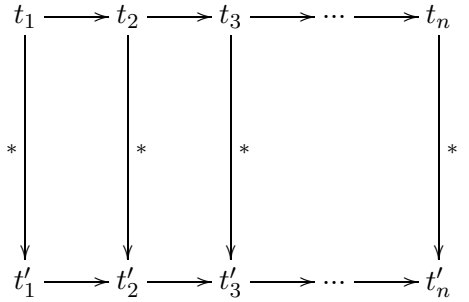


where the first square comes into two flavors depending on whether the rewrite downward is a labelled or not. Both cases, however, are covered by the lemma above. Removing the labelling gives:

Lemma 6.5 (Strip lemma) *In an orthogonal left-linear system every divergence of a one-step rewriting against a multi-step rewriting has a convergence*



The main theorem now follows easily from this lemma by pasting together the strips to obtain a convergence for a divergence in which the horizontal rewrite is multi-steps as well.



7 Standard reductions

Recall that in the λ-calculus even when one has confluence that one must be careful about the order in which one reduces if one wants to be assured of finding a normal form.

We shall say that a reduction path

$$t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow .... \longrightarrow t_n$$

for an orthogonal left-linear system is **standard** if it is never the case that for $t_i \xrightarrow{r_i} t_{i+1} \xrightarrow{r_{i+1}} t_{i+2}$ the redex of r_{i+1} is a above that of r_i. Notice this means that the redex for r_{i+1} must already be present in t_i and we can, in fact, swap the reduction order:

$$\begin{array}{ccc} t_i = t(r_1(t_1,..,t_j(r_2(s_1,...,s_m)),...,t_n) & \xrightarrow{\quad r_2 \quad} & t(r_1(t_1,..,t_j(c_2[s_1/y_1,...,s_m/y_m]),...,t_n)) \\ \Big\downarrow{\scriptstyle r_1} & & \Big\downarrow{\scriptstyle r_1} \\ t(c_1[t_1/x_1,...,t_j(r_2(s_1,...,s_m))/x_i,...x_n/t_n]) & \xrightarrow[r_2]{*} & t(c_1[t_1/x_1,...,t_j(c_2[s_1/y_1,...,s_m/y_m])/x_i,...x_n/t_n]) \end{array}$$

Note that the left-linearity is used here and allow us to deal with an arbitrary pair of reductions. Putting the reductions in standard order can increase the number of reductions (when the variable x_i occurs more than once in c_1) but also, importantly, it can reduce the number of reductions (especially when there are infinitely many possible reductions from $t_j(c_2[s_1/y_1,...,s_m/y_m])$) as the variable x_i may not occur in c_i.

Notice that in the modified reduction chain that the maximal number of sequential reductions never increases. Two reductions are sequential (as opposed to parallel) when one occurs in a subterm of the other. This means the process of standardizing a reduction sequence will always terminate.

Proposition 7.1 *In a left-linear orthogonal rewriting system*

(i) *every reduction sequence can be transformed into a standard reduction sequence;*

(ii) *If a term has a normal form it can be obtained by a standard reduction.*

The cost of doing a standard reduction is apparently that one may have to do more reductions. However, notice that the increase in the number of reductions can be *completely* mitigated if one holds the term so that the substitutions do not replicate rewrites. This is usually done by regarding the terms as graphs so that subterms can be shared and, more importantly, the reductions on that subterm are shared (so only done once). This is essentially the idea behind graph reduction. Formally this means that these left-linear systems have an optimal rewriting strategy.

Notice that a standard reduction does not indicate which redex should be reduced first and, indeed, beyond saying that the outermost reductions should *all* be done first it cannot offer more guidance (this is the parallel outermost strategy). Here is an example of a linear rewrite system (due to Huet) where a leftmost outermost strategy will fail:

$$\begin{array}{rcl} f(x,b()) & \longrightarrow & d() \\ a() & \longrightarrow & b() \\ c() & \longrightarrow & c() \end{array}$$

Note that $f(c(), a())$ has a normal form but a leftmost outermost strategy will not find it!

8 Combinatory algebra

As mentioned above a combinatory algebra is a model for an algebraic system with a binary operation, called application, and two constants k and s. These are subject to the following equations which we orient as rewrites:

$$\begin{array}{rcl}(\mathsf{k} \bullet x) \bullet y & \longrightarrow & x \\ ((\mathsf{s} \bullet x) \bullet y) \bullet z) & \longrightarrow & ((x \bullet z) \bullet (y \bullet z)\end{array}$$

As we have seen above these rewrite rules form an orthogonal left-linear rewriting system. Therefore, this is a confluent system in which standard reductions are guaranteed to find normal forms. It is also a non-trivial system as $\mathsf{s}\bullet\mathsf{k} \neq \mathsf{k}$. The system, in fact, satisfies some quite remarkable properties which we now briefly explore.

An **applicative system** is a set A with a single binary operation

$$_ \bullet _ : A \times A \longrightarrow A; (x, y) \mapsto x \bullet y.$$

Clearly a combinator algebra is an example of a applicative system as one can simply forget about k and s. An applicative system is said to be functionally complete if whenever there is a polynomial expression $p(x_1, ..., x_n)$ in n variable

$$p \longrightarrow x_1 | x_2 | ... | x_n | a \in A | p \bullet p$$

then there is an element $\widehat{p}$ such that for every substitution of the variables by elements of A

$$(...((\widehat{p} \bullet x_1) \bullet ...) \bullet x_n = p(x_1, ..., x_n)$$

Proposition 8.1 *Combinatory algebra is functionally complete. Furthermore, a combinatory complete applicative system is a combinatory algebra*

Proof: The proof involves the introduction of an abstraction mechanism which is a mechanism for building constants with the desired property for functional completeness. Suppose, therefore, we have a polynomial p which involves free variables $X = x_1, ..., x_n$ and then define $\lambda^* x_i.p$ as the following polynomial in which the variable x_i has been removed:

$$\begin{array}{rcll}\lambda^* x.x & = & (\mathsf{s} \bullet \mathsf{k}) \bullet \mathsf{k} & \\ \lambda^* x.z & = & \mathsf{k} \bullet z & z \text{ is } \mathsf{s} \text{ or } \mathsf{k} \text{ or a variable} \\ \lambda^* x.(p_1 \bullet p_2) & = & (\mathsf{s} \bullet (\lambda^* x.p_1)) \bullet (\lambda^* x.p_2) & \end{array}$$

Observe that $(\lambda * x.p) \bullet M = p[M/x]$ which can be seen by a structural induction:

$$\begin{array}{rcl}(\lambda^* x.x) \bullet M & = & ((\mathsf{s} \bullet \mathsf{k}) \bullet \mathsf{k}) \bullet M \\ & = & (\mathsf{k}) \bullet M) \bullet (\mathsf{k}) \bullet M) = M = x[M/x] \\ (\lambda^* x.z) \bullet M & = & (\mathsf{k} \bullet z) \bullet M = z = z[M/x] \\ \lambda^* x.(p_1 \bullet p_2) \bullet M & = & ((\mathsf{s} \bullet (\lambda^* x.p_1)) \bullet (\lambda^* x.p_2)) \bullet M \\ & = & ((\lambda^* x.p_1) \bullet M) \bullet ((\lambda^* x.p_2) \bullet M) \\ & = & p_1[M/x] \bullet p_2[M/x] = (p_1 \bullet p_2)[M/x]\end{array}$$

Also we observe that $(\lambda^* x.p)[M/y] = (\lambda^* x.p[M/y])$ when $x \notin FV(p)$ again a proof by structural induction is required which we leave to the reader). This now shows that we can take

$$\widehat{p} := (\lambda^* x_1.(\lambda^* x_2....(\lambda^* x_n.p)...)$$

to obtain:

$$\begin{array}{rcl} ((((\lambda^* x_1.(\lambda^* x_2....(\lambda^* x_n.p)...)) \bullet M_1) \bullet M_2) \bullet ...) \bullet M_n \\ = & (((\lambda^* x_2.(...(\lambda^* x_n.p)...)[M_1/x_1]) \bullet M_2) \bullet ...) \bullet M_n \\ = & ((\lambda^* x_2.(...(\lambda^* x_n.p[M_1/x_1])...) \bullet M_2) \bullet ...) \bullet M_n \\ = & p[M_1/x_1, ...M_n/x_n] \end{array}$$

For the converse it suffices to show that a combinatory algebra which is functionally complete has a k and an s. However, the equations for these combinators are functional completeness equations so this must be so! □

These observations suggest that in combinatory algebra, following the λ-calculus

$$\Omega = (\lambda^* x.x \bullet x) \bullet (\lambda^* x.x \bullet x)$$

should have a non-terminating reduction. This is easily checked to be the case. However, it should not be imagined that the two systems are equivalent: in combinatory algebra:

$$N = M \not\Rightarrow \lambda^* x.M = \lambda^* x.N$$

Thus $(\mathsf{k} \bullet x) \bullet y = x$ but

$$\begin{array}{rcl} \lambda^* x.(\mathsf{k} \bullet x) \bullet y & = & (\mathsf{s} \bullet (\lambda^* x.\mathsf{k} \bullet x)) \bullet (\lambda^* x.y) \\ & = & (\mathsf{s} \bullet ((\mathsf{s} \bullet (\lambda^* x.\mathsf{k})) \bullet (\lambda^* x.x))) \bullet (\lambda^* x.y) \\ & = & (\mathsf{s} \bullet ((\mathsf{s} \bullet (\mathsf{k} \bullet \mathsf{k})) \bullet ((\mathsf{s} \bullet \mathsf{k}) \bullet \mathsf{k}))) \bullet (\mathsf{k} \bullet y) \\ & \neq & (\mathsf{s} \bullet \mathsf{k}) \bullet \mathsf{k} = \lambda^* x.x \end{array}$$

So that combinatory logic is much weaker. Combinatory logic is important as a system as it one of the simplest systems in which all (partial) computable functions can be simulated. The encoding technique follows the techniques of the λ-calculus.