*URL:* [*http://www.elsevier.nl/locate/entcs/volume57.html*](http://www.elsevier.nl/locate/entcs/volume57.html) *23 pages*

*Just-in-time: on Strategy Annotations*

*Jaco van de Pol*

*Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands Email:* [*Jaco.van.de.Pol@cwi.nl*](mailto:Jaco.van.de.Pol@cwi.nl)

*Abstract*

*A simple kind of strategy annotations is investigated, giving rise to a class of strate- gies, including leftmost-innermost. It is shown that under certain restrictions on annotations, an interpreter can be written which computes a normal form of a term in a bottom-up traversal. The main contribution is a correctness proof of this inter- preter. Furthermore, a default strategy annotation is provided, called just-in-time, which satis es the criteria for the interpreter. The just-in-time strategy has a better termination behaviour than innermost rewriting for many interesting examples.*

## *1 Introduction*

*A term rewrite system (TRS) is a set of directed equations. A term is evaluated* by repeatedly replacing a subterm that is an instance of the left-hand side of an equation (a redex) by the corresponding instance of the right-hand side (the contractum) until a term is reached which contains no redex (a normal form). Because a term can have many redexes, an implementation has to follow a certain strategy, that tells at any moment which redex should be chosen. A strategy is often chosen for its eÆciency: following a good strategy results in short rewrite sequences. A smart strategy may even avoid in nite computations.

*For an actual interpreter, one must also take into account the cost of*

*nding the next redex. This is the reason that many systems implement* leftmost-innermost rewriting, although this may produce relatively long re- duction sequences, and has a bad termination behaviour [[7](#_bookmark18)]. We will use annotations as a simple way to specify strategies.

*1.1 Strategy Annotations*

*Consider the following term rewrite system (TRS), where if , T and F are* function symbols, b, x, y are variables, and which has three rewrite rules,

*c 2001 Published by Elsevier Science B. V. Open access under* [*CC BY-NC-ND license.*](http://creativecommons.org/licenses/by-nc-nd/3.0/)

*named , and :*

*: if (T; x; y) 7! x*

*: if (F; x; y) 7! y*

*: if (b; x; x) 7! x*

*A natural way of normalizing the term if (p; q; r) is to rst normalize p, and* then try rule or . This procedure avoids unnecessary reductions inside q or r. In some cases, this could even prevent non-terminating computations. If the rst argument doesn't reduce to T or F (for instance because p is an open term or because some rules are missing), the second and third argument must be normalized, and nally the last rule is tried. The sketched procedure can be very concisely represented by the following strategy annotation for if :

*strat(if ) = [1; ; ; 2; 3; ]:*

*We say that rule and need the rst argument, because they match* on it. Rule needs the second and third arguments, because it compares them. We say that the annotation is in-time because the arguments of if are evaluated before the rules which need them are tried. We say that this annotation is full because all argument positions and rules for if occur in it. Another full and in-time annotation for if would be [1; 2; 3; ; ; ], de- noting the left-most innermost strategy. We will de ne a default annotation, which evaluates all its arguments from left to right, and tries to apply a rule as soon as its needed arguments are evaluated. We call this default strategy

*the just-in-time strategy. Note that strat (if ) is the just-in-time strategy.*

*1.2 Contribution*

*Following* [*[18*](#_bookmark27)*], we de ne a normalizing function norm(t), which normalizes a* term according to a strategy annotation for all function symbols. It traverses the term once, and computes a normal form in a bottom-up fashion. If a redex is found at a certain position, it is replaced and the search proceeds at the same position. So for these strategy annotations, nding the next redex is as eÆcient as for innermost rewriting. The normalization function has been used as a design to build an actual interpreter in the programming language C.

*Viewing normal forms as the correct answers, partial correctness of the* normalizer means that if norm(t) = s, then s is a normal form of t. We call a strategy annotation complete if norm is partially correct. Our main result is that full and in-time are suÆcient syntactic criteria for completeness. This result applies to any TRS, without restrictions. This generalizes [[17](#_bookmark23)], because our restrictions are more liberal, and that proof only works for left-linear TRSs.

*The proof yields some extra information: although norm continues at the* position where the previous redex was found, it is equivalent to a particular memory-less strategy. This means that the chosen redex depends on the term

*only, and not on previous reduction steps. Moreover, norm(t) follows this* strategy even for in nite reduction sequences.

*A default strategy annotation (called just-in-time) that is full and in-time* can be computed automatically, and is satisfactory for many function def- initions, including if-then-else and the boolean connectives conjunction and disjunction.

*1.3 Related Work*

*Many rewrite (logic) implementations allow the user to specify a strategy.* ELAN [[2]](#_bookmark12) was the rst rewrite implementation, where users can de ne their own strategies. Rewrite rules are viewed as basic strategies, that can be composed by sequential, alternative and conditional composition. Mechanisms to control non-determinism are also present. In Maude [[5]](#_bookmark16) strategies can be de ned inside the logic, thanks to the re ection principle of rewrite logic. Stratego [[19]](#_bookmark28) incorporates recursive strategies and general traversal patterns. It was shown [[14](#_bookmark24)] how these can be de ned inside ASF+SDF [[4,3].](#_bookmark14)

*All mentioned strategy languages are far richer than the annotations stud-* ied in our paper. Those systems advocate a separation between computations (rules) and control (strategies). By writing strategies the user can freely choose when the rules are applied. Important applications are the speci cation of pro- gram transformations. However, these strategies are not always complete, in the sense that a strategy might terminate in a term that still contains redexes. As far as we know, no analysis exists whether subclasses of these strategies are complete, which is an important issue if one is interested in nding normal forms.

*Members of the OBJ-family* [*[9,16,18*](#_bookmark27)*] have strategy annotations that are* similar to the ones discussed in our paper. In OBJ an annotation is a list of integers. Similar to us, +i denotes the normalization of the i-th argument. There are two di erences. First, instead of mentioning rules individually (our

*, ), OBJ uses 0 to denote a reduction at top level with any rule. Our*

*more re ned notion allows to assign a priority in applying the rules. Avoiding* repetitions of 0 (which would lead to multiple calls to the matching procedure), in OBJ the only full and in-time annotation for the three if -rules mentioned before is the strategy [1; 2; 3; 0], which corresponds to innermost rewriting. The second di erence is that OBJ allows i, denoting that argument i is only normalized on demand (i.e. if it is needed for matching with another rule). Such annotations specify lazy rewriting, which we have not studied.

*The default strategy of CafeOBJ is similar to our just-in-time annotation.* We cite from [[16,](#_bookmark25) p. 83]: For each argument, evaluate the argument before the whole term, if there is a rewrite rule that de nes the operator such that, in the place of the argument, a non-variable term appears. We added to this: \or if in the place of the argument, a non-left-linear variable occurs". This extra condition is necessary for obtaining the completeness result of our paper. It is

*not clear from* [*[9,16]*](#_bookmark25) *whether the OBJ-systems check the completeness of the* user-provided annotation.

*In* [*[15]*](#_bookmark26) *and* [*[17]*](#_bookmark23) *completeness of OBJ-annotations is studied.* [*[15]*](#_bookmark26) *only* considers eager annotations, and proves (Thm. 6.1.12) that full annotations ending in a 0 are complete. This is generalized in [[17,](#_bookmark23) Cor. 3.8] by allowing argument positions after the last 0. In a separate correction, those authors indicate that their proof doesn't work for non-left-linear TRSs. Moreover, in [[17](#_bookmark23)] the criteria depend on all occurrences of function symbols in left hand sides, where our criteria only depend on head-occurrences. As a consequence, given the left hand sides g(f (c)) and f (x), the annotation f : [0; 1] is not allowed by the criteria of [[17],](#_bookmark23) where it would be allowed by our criteria. So we generalize the mentioned results, by having more liberal criteria on a larger class of TRSs. On the other hand, [[17]](#_bookmark23) also considers criteria for on-demand annotations, which we have not studied.

*All normalization functions we found in the literature, e.g.* [*[18,15*](#_bookmark26)[*,17]*](#_bookmark23) *are* presented with some memory (either by labeling or by using non-tail-recursive calls). Our correctness proof provides the extra information that the inter- preter actually follows a certain memory-less strategy, even in case of diver- gence.

*In* [*[11*](#_bookmark22)*] a survey of strategies in term rewriting is given. The focus is on* normalizing strategies for orthogonal systems. A strategy is normalizing if it

*nds a normal form whenever one exists. Orthogonality is a syntactic criterion* which ensures con uence. For non-orthogonal systems only few results on normalizing strategies exist. We have not studied which class of annotations gives rise to normalizing strategies. On the other hand, our results apply to non-orthogonal term rewriting systems as well. In [[15](#_bookmark26)] decidable criteria on eager OBJ-annotations are provided which are suÆcient to ensure normalizing strategies for orthogonal TRSs. These results can probably be adapted to our annotations. Recently, [[12,13]](#_bookmark24) studied normalizability of positive OBJ strategies and of our strategy annotations and termination of rewrite systems using such strategies. These issues are studied in the framework of context sensitive rewriting.

## *2 Basic De nitions and Result*

*2.1 Preliminaries*

*We take standard de nitions from term rewriting* [*[11,1].*](#_bookmark15) *We presuppose a* set of variables, and function symbols (f ), each expecting a xed number of arguments, denoted by arity(f ). Terms are either variables (x) or a function symbol f applied to n terms, denoted f (t1;::: ; tn), where n is the arity of f . With head (t) we denote the topmost function symbol of t.

*A position (p, q) is a string of integers. By " we denote the empty string.* With tjp we denote the subterm of t at position p, which is only well-de ned

*if p is a position in t (see* [*[1]*](#_bookmark15) *for a formal de nition). In that case, tj" = t and* f (t1;::: ; tn)ji:p = tijp. With t[s]p we denote the term t in which tjp is replaced by s. We write p q if p is a pre x of q (i.e. p:p0 = q for some p0 ).

*A rewrite rule is a pair of terms l 7! r, where l is not a variable, and all* variables occurring in r occur in l as well. A term rewrite system (TRS) is a set of rewrite rules. A substitution is a mapping from variables to terms, and with t we denote the term t with all variables x replaced by (x). A TRS R induces a rewrite relation on terms as follows: t !R t[r]p if and only if tjp = l for some rule l 7! r 2 R. In this case l is called the redex and r the contractum, and the pair (l; r) is called a rewrite in [[11](#_bookmark22)]. A normal form is a term t which contains no redex. Note that a redex may have many occurrences in the same term, so in order to uniquely identify the rewrite step, we also need a position p. From the position p the redex l can be reconstructed. For this reason, it is convenient to call the pair (p; r) a rewrite of t.

*2.2 Strategy Annotations and Strategies*

*A strategy annotation for a function symbol f in TRS R is a list whose elements* can be either:

*a number i, with 1 i arity(f ); or*

*a rule l 7! r 2 R, such that head (l) = f .*

*Without loss of generality, we will assume that an annotation has no dupli-* cates, i.e. each i occurs at most once (after the rst normalization the i'th argument is normal, so a second occurrence of i would not contribute an ac- tual rewrite step). We write [] for the empty annotation and [xjL] for the annotation with head x and tail L. In the sequel i; j; k will range over argu- ment positions, and ; ; over rewrite rules. So [ijL] starts with a natural number and [ jL] with a rewrite rule.

*An index i is needed for a left hand side f (l1;::: ; ln), if li is not a variable,* or if it is a variable which occurs in lj, for some j 6= i. Index i is needed for rule : l 7! r if i is needed for l. A strategy annotation L is full for f , if for each i with 1 i arity(f ), i 2 L and for each rule : l 7! r 2 R with head (l) = f , 2 L. A strategy annotation L is in-time, if for any and i such that L = L1 L2 iL3 , i is not needed for . In a full and in-time annotation all needed positions for occur before . The distinguishing feature of the notion `needed' is as follows:

*Lemma 2.1 Let l = f (l1 ;::: ; ln) and let argument i be not needed for l. If* t = l for some , then for any s, t[s]i = l for some .

*Proof. For some , t = l = f (l;::: ; l). Because i is not needed for l, li is*

*1 n*

*a variable. Let = [li := s]. As i is not needed, li doesn't occur in lj (for*

*j 6= i), so l = l, and l = s. Hence l = t[s] .* *2*

*j j i* *i*

*We now de ne the strategy associated to a strategy annotation. A strategy* can be viewed [1](#_bookmark2) as a partial function that given a term t, yields some rewrite of t, i.e. a pair (q; s) such that tjq = l and s = r for some rule l 7! r and substitution . In this case t !R t[s]q. Alternatively the function may yield

*? (unde ned { found no rewrite). A complete strategy yields ? on t only if t*

*is a normal form.*

*In the sequel, a xed TRS R is supposed, with a xed strategy annotation* strat . We write strat (t) as an abbreviation of strat (head (t)), i.e. the strategy annotation of its head symbol, where strat (x) = [] for variables x. We say that strat is full (in-time) if strat (f ) is full (in-time) for all symbols f . Next we de-

*ne rewr1 (t; L), which computes the next rewrite in t according to annotation*

*L. We allow a slight overloading as in rewr1 (t) in case L = strat (t).* De nition 2.2 rewr1 (t) = rewr1 (t; strat(t)), where:

*rewr1 (t; []) = ?*

# *>8*

*rewr1 (t; [l 7! rjL]) =*

*<>*

# *:>>8*

*rewr1 (t; [ijL]) = ><>*

*:>>*

*if t = l for some then ("; r)*

*else rewr1 (t; L)*

*if rewr1 (tji) = (q; s) for some q, s* then (i:q; s)

*else rewr1 (t; L)*

*The de nition proceeds by induction on t and the strategy-annotation L.* In each recursive call, either the term t gets smaller, or it remains equal and the list L gets smaller. Therefore this function terminates either in (q; s) or in

*?. We now show that for full annotations, the associated strategy is complete:*

*Proposition 2.3 If strat is full and rewr1 (t) = ?, then t is a normal form*

*Proof. The proof is with induction on t. Assume that t is not in normal* form. Then it contains a redex, either at top level, or in a proper subterm. We distinguish these two cases:

*Assume that t = l for some rule : l 7! r. Then, by induction on L one* can show: \if 2 L then rewr1 (t; L) is de ned". By fullness, 2 strat (t), so rewr1 (t) = rewr1 (t; strat (t)) is de ned.

*Assume that tji contains a redex. Then using the induction hypothesis, one* can show with induction on L: \if i 2 L, then rewr1 (t; L) is de ned". By fullness, i 2 strat (t), so rewr1 (t) = rewr1 (t; strat (t)) is de ned. *2*

*1 This covers deterministic, one-step, memory-less strategies only.*

*2.3 Problem Statement*

*Given a strategy annotation, the associated reduction sequence can be de ned* as follows:

# *8*

*seq1 (t) = <>*

*>*

*:>>*

*if rewr1 (t) = (q; s) for some q, s then t :: seq1(t[s]q)*

*else hti*

*By the previous proposition, a normal form can be obtained as last(seq 1(t))* (for in nite sequences this is unde ned). The computational drawback is that after each step the whole term t[s]q must be traversed to nd the next rewrite. This repeats a lot of work of the previous step. It would be nice if the search could be continued at position q. Therefore we propose the following partial function, norm(t; L), which tries to nd a normal form of t, according to annotation L. We allow a slight overloading, as in norm(t), in case L is the

*xed strategy annotation strat . We view this function as the design for an* interpreter.

*De nition 2.4 norm(t) = norm(t; strat (t)), where:*

*norm(t; []) = t*

# *8*

*><>*

*norm(t; [l 7! rjL]) =*

*:>>*

*if t = l for some then norm(r)*

*else norm(t; L)*

*norm(t; [ijL]) = norm(t[norm(tji)]i; L)*

*Avoiding position-notation, the last clause can be written alternatively as* norm(f (t1 ;::: ; ti;::: ; tn); [ijL]) = norm(f (t1 ;::: ; norm(ti);::: ; tn); L).

*If t is a non-terminating term, norm(t) might diverge, in which case it is* unde ned. The next section is devoted to the technical core of this paper, viz. correctness of norm. That is, we must prove

*Theorem 2.5 If strat is in-time, then norm(t) = last(seq 1 (t)).*

*This follows immediately from Propositions* [*3.1*](#_bookmark4) *and* [*3.7.*](#_bookmark9) *In combination with* Proposition [2.3,](#_bookmark1) we obtain that for full and in-time strategies, if norm(t) = s, then s is a normal form.

*2.4 Counter examples*

*It may be illustrative to show why the conditions on annotations are needed.* Consider the system with three if -rules from the introduction, and an addi- tional rule T ^ T 7! T .

*The strategy-annotation [ ; ; 1; 2; 3; ] is not in-time, because matches* on the rst argument. Consider the term if (T ^ T; x; y). Rule and are not

*immediately applicable. After reduction of T ^ T , and will not be tried* again. So under this annotation, norm(if (T ^ T; x; y)) = if (T; x; y), which is not normal. Similarly, [1; ; ; ; 2; 3] is not in-time, because is non-linear in its second and third argument. Under this annotation, norm(if (x; T ^T; T )) = if (x; T; T ). This is not normal, due to the fact that was tried too early. Finally, [ ; ; 2; 3; ] is not full, because argument position 1 is missing. Under this annotation norm(if (T ^ T; x; y)) = if (T ^ T; x; y), which is not normal.

*These examples show that the conditions cannot be dropped in general.* In certain cases they could be weakened. For instance in : f (x) 7! g(x), the annotation [ ] is not full, but this is harmless because applies to any term with head symbol f . This weakening is inessential, because the behaviour of the interpreter is exactly the same as with the full strategy [ ; 1].

## *3 Correctness Proof*

*The proof has two distinct parts. First we identify the series of redexes con-* tracted by norm. This is not straightforward due to its doubly recursive de nition. By program transformation we nd an equivalent function norm2, where the double recursion is eliminated in favour of a stack containing the return points. From this de nition the series of redexes can be easily extracted (Section [3.1).](#_bookmark3)

*At rst sight, norm doesn't follow a memory-less strategy, because after*

*nding a redex at position q, it continues its search from q onwards. Therefore,* the found rewrite depends on a certain internal \state" or \memory", say S. Hence the strategy will be a function of the form rewr2 (t; S) = (q; s; R), where (q; s) is the found rewrite, and R denotes the next state. In the sequel, the triple (q; s; R) will also be called a rewrite.

*The second step in the proof (Section* [*3.2)*](#_bookmark5) *shows that if the annotation is* in-time, then the state doesn't in uence the rewrite found. That is, the next rewrite can be found in two equivalent ways: rewr2 (t[s]q; S) = rewr2 (t[s]q; R). The proof is then nished by the observation that for the initial state I, rewr2 (t; I) = rewr1 (t).

*3.1 Making Recursion Explicit*

*This section eliminates the double recursion from norm. In the rst trans-* formation, we replace recursion on subterms by recursion on positions. This makes it possible to return to a previous stage. First specify:

*norm1(t; p; L) = t[norm(tjp; L)]p*

*Next, using this de nition and the de ning equations for norm, we can calcu-* late (Section [A.1)](#_bookmark29) the following recursive de nition for norm1 :

*norm1(t; p; []) = t*

# *8*

*><>*

*norm1(t; p; [l 7! rjL]) =*

*:>>*

*if tjp = l for some*

*then norm1(t[r]p; p; strat(r)) else norm1 (t; p; L)*

*norm1 (t; p; [ijL]) = norm1(norm1 (t; p:i; strat(tjp:i)); p; L)*

*Next, we eliminate the double recursion in favour of a stack, which is a* list of pairs of previous positions and the annotations that still have to be executed. To this end, we introduce the recursive speci cation for norm2:

*norm2 (t; []) = t*

*norm2(t; [(p; L)jS]) = norm2(norm1 (t; p; L); S)*

*From this speci cation, and the recursive equations derived for norm1, we can* derive (Section [A.2)](#_bookmark30) the following recursive equations for norm2 :

*norm2 (t; []) = t*

*norm2 (t; [(p; [])jS]) = norm2(t; S)*

*8> if tjp = l for some*

*norm2 (t; [(p; [l 7! rjL])jS]) = ><*

*>>:*

*then norm2(t[r]p; [(p; strat(r))jS]) else norm2 (t; [(p; L)jS])*

*norm2(t; [(p; [ijL])jS]) = norm2(t; [(p:i; strat (tjp:i)); (p; L)jS])*

*From this explicit de nition it is easy to guess the next rewrite that norm2* will take, given the current state (stack) S. This gives rise to the following de nition rewr2 (t; S). The result will be either ?, or a triple (q; s; T ), where (q; s) denotes the rewrite as previously, and T is the stack after replacing t[s]q.

*rewr2 (t; []) = ?*

*rewr2 (t; [(p; [])jS]) = rewr2 (t; S)*

# *8*

*><>*

*rewr2 (t; [(p; [l 7! rjL])jS]) =*

*:>>*

*if tjp = l for some*

*then (p; r; [(p; strat(r))jS])*

*else rewr2 (t; [(p; L)jS])*

*rewr2 (t; [(p; [ijL])jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L)jS])*

*Given this function rewr2 , we can de ne a second rewrite sequence. This* time the sequence is not memory-less, because each step changes the state

*(stack S) of the system.*

*8> if rewr2 (t; S) = (q; s; R) for some q, s, R*

*seq2(t; S) = ><*

*>>:*

*then t :: seq2 (t[s]q; R) else hti*

*In order to check that rewr2 (t; S) indeed yields the next rewrite taken by* norm2 in state S, one can take the speci cation

*norm3 (t; S) = last(seq2 (t; S))*

*Using the de nitions of seq 2 and rewr2 , one can derive recursive equations for* norm3 (Section [A.3),](#_bookmark31) which appear to be exactly the same as those for norm2. We summarize the result of this section:

*Proposition 3.1 norm(t; strat (t)) = last(seq 2(t; [("; strat (t))])).*

*Proof. First, norm1(t; p; L) = norm2 (norm1 (t; p; L); []) = norm2(t; [(p; L)]).* Also, norm(t; L) = t[norm(tj"; L)]" = norm1(t; "; L). The result follows be- cause by the previous remark norm2(t; S) = last(seq 2(t; S)). *2*

*3.2 Connecting Memory-less and State-based Strategies*

*It is now suÆcient to prove that seq2 (t; [("; strat(t))]) = seq 1(t). This is the* case if rewr2 (t; S) yields the same rewrite as rewr1 (t), for all reachable states S. To this end, we rst show that the stack will be always well-formed (de-

*ned below). Then we show that in fact rewr2 is actually independent of the* current state, i.e. rewr2 (t; S) = rewr2 (t; [("; strat (t))]) for all stacks S encoun- tered (Lemma [3.5).](#_bookmark8) Finally, we show that rewr2 (t; [("; strat(t))]) = rewr1 (t) (Lemma [3.6).](#_bookmark10)

*We now de ne the set of well-formed stacks w.r.t. t. Intuitively, the po-* sitions in the stack form a proper path in t, all annotations on the stack are in-time, and nodes can be visited at most once.

*De nition 3.2 The set of well-formed stacks w.r.t. t are de ned inductively* as follows:

*[] is well-formed.*

*[("; L)] is well-formed, if L is an in-time strategy annotation for head (t).*

*[(p:i; K); (p; L)jS] is well-formed, if [(p; L)jS] is well-formed, and p:i is a* position in t and K is an in-time strategy annotation for head (tjp:i) and i 62 L.

*Lemma 3.3 If strat is in-time, then it is an invariant of rewr2 (and norm2* and seq2 ) that the stack is well-formed.

*Proof. [("; strat (t))] is well-formed (initial condition) and the property is pre-* served in all recursive calls. This relies on the assumption that strategy an- notations contain no duplicates. *2*

*Lemma* [*3.4.3*](#_bookmark7)

*Lemma* [*3.4.2*](#_bookmark7)

*p*

*q*

*p*

*q*

*s*

*Lemma* [*3.4.1*](#_bookmark7)

*Finding a redex in t*

*Finding a redex in t[s]q*

*Fig. 1. The search in t can be mimicked in t[s]q.*

*The following technical lemma is the core of the proof. It shows that if search-* ing in t from state S yields a rewrite (q; s; R), then the search in t from S can be mimicked in t[s]q, and will lead again to state R (see Figure [1).](#_bookmark6) The key of the proof is that if rule is not applicable at tjp, then a reduction inside tjp can only occur in an argument which is not needed, so also in t[s]qjp rule will not be applicable. The full proofs of Lemma [3.4{3.7](#_bookmark9) are in Appendix [B.](#_bookmark32)

*Lemma 3.4 Let strat be in-time. Let [(p; L)jS] be a well-formed stack. As-* sume rewr2 (t; [(p; L)jS]) = (q; s; R), for some q, s and R. Then we have:

*(i) If q 6 p then rewr2 (t; [(p; L)jS]) = rewr2 (t; S).*

*(ii) If p = q then R = [(p; strat(s))jS].*

*(iii) If q 6 p, then rewr2 (t[s]q; [(p; L)jS]) = rewr2 (t[s]q; R).*

*Proof sketch.*

1. *Induction on the structure of tjp, and for equal tjp on the structure of L. The proof proceeds by case distinction on L.*
2. *Induction on L, using (i) in case L = [ijL0 ].*
3. *Starting with stack [(p; L)jS] and term t, rewr2 reduces in a number of steps to (q; s; R). The proof proceeds by mimicking this reduction starting with the same stack in term t[s]q. The proof is by induction on the number of recursive calls of rewr2 (t; [(p; L)jS]) to (q; s; R).* *2*

*Lemma 3.5 Let strat be in-time. If rewr2 (t; [("; strat(t))]) = (q; s; R) then* rewr2 (t[s]q; R) = rewr2 (t[s]q; [("; strat(t[s]q))]).

*Proof sketch. If q = ", this follows from Lemma* [*3.4.(ii).*](#_bookmark7) *Otherwise, q > ",* and the result follows from Lemma [3.4.(iii).](#_bookmark7) *2*

*Finally, we prove the relationship between rewr1 and rewr2 :* Lemma *3.6*

1. *rewr1 (t) = (q; s) () for some R, rewr2 (t; [("; strat(t))]) = (q; s; R)*
2. *rewr1 (t) = ? () rewr2 (t; [("; strat(t))]) = ?*

*Proof sketch. The proof follows from the following propositions, which can* be proved by simultaneous induction on the structure of tjp and, for equal tjp, on L.

1. *if rewr1 (tjp; L) = (q; s), then for some R, rewr2 (t; [(p; L)jS]) = (p:q; s; R).*
2. *if rewr1 (tjp; L) = ?, then rewr2 (t; [(p; L)jS]) = rewr2 (t; S).* *2*

*Proposition 3.7 If strat is in-time, then seq2(t; [("; strat (t))]) = seq1 (t).*

*Proof sketch. Using Lemma* [*3.5*](#_bookmark8) *and* [*3.6*](#_bookmark10) *the de nition of seq 2(t; [("; strat(t))])* can be transformed into the de nition of seq1 (t). *2*

## *4 Implementation and Applications*

*We have constructed a C-implementation of the function norm, which acts* as an interpreter of a given TRS annotated by some strategy. As a default, the system computes the just-in-time strategy during initialization. We rst describe this annotation, and then mention some implementation issues.

*4.1 The Just-in-time Strategy Annotation*

*The just-in-time strategy is de ned as follows. For any function symbol f ,* with arity n, take the list [1;::: ; n]. Next, insert each rule directly after the last argument position that it needs (due to matching or non-linearity). If several rules are placed between i and i + 1, the textual order of the original speci cation is maintained.

*We applied this strategy to several speci cations, with satisfactory results.* The application domain is veri cation of distributed systems, where a system speci cation has a process part and an algebraic data speci cation part. A theorem prover is being implemented, to solve boolean combinations of equal- ities over the algebraic data speci cation, by a combination of BDDs and term rewriting, along the lines of [[10].](#_bookmark21) In many cases, innermost rewriting didn't lead to a normal form. Below we list a number of rules in order to illustrate

*this point:*

*: F \_ x 7! x*

*: T \_ x 7! T*

*: count(l) 7! if (empty(l); 0; 1+ count(tail(l)))*

*Æ : div(m; n) 7! if (m < n; 0; 1+ div(m n; n))*

*: rem(m; n) 7! if (m < n; m; rem(m n; n))*

*On closed lists, count terminates with the just-in-time annotation [ ; 1] (as-* suming standard de nitions of empty and tail), but it diverges with innermost rewriting. This could be solved by replacing if by pattern matching, provid- ing rules for count([]) and count([xjL]). However, this solution is not easily available for div and rem (division and remainder). Assuming standard de - nitions of and <, these functions terminate on closed numerals m and n for positive n, with the just-in-time annotation [Æ= ; 1; 2], but they diverge with innermost rewriting. The just-in-time annotation for \_ is [1; ; ; 2]. With this annotation, eq(n; 0) \_ div(m; n) < m terminates for all numerals m and n, even for n = 0, provided eq(0; 0) ! T .

*The just-in-time strategy works from left-to-right. Sometimes it is more* eÆcient to start with another argument. One could devise an algorithm to transform a TRS by reordering the arguments to the function symbols. In general, one could study which of the full and in-time annotations yields the most eÆcient strategy for a given TRS.

*4.2 Implementation Issues*

*We now shortly mention some well-known implementation issues. First, a* rule : f (x) 7! g(x; x; x) with annotation [ ; 1], would copy all redexes in x three times. Therefore, in our implementation we use maximally shared terms (DAGs), in which x occurs only once. The implementation uses the ef-

*cient annotated term library* [*[3],*](#_bookmark14) *providing maximally shared terms, garbage*

*collection and term tables (for memoization) for free.*

*Another issue is that in a rule : f (x) 7! g(x) with annotation [1; ], rst* x is normalized by f to n, and then g(n) is called. Because g doesn't know that n is normal, it will traverse the whole n. To avoid this, all subterms which are known to be normal are marked. So g will get a marked argument, which it doesn't traverse. If the annotation would be [ ; 1], as with the just-in-time strategy, g would get x unmarked. A similar approach can be found in [[18](#_bookmark27)].

*Finally, consider the rule : f (x) 7! g(h(x)). In innermost rewriting,* *x* can be normalized, passing the result to function f . Then f calls function h and g, respectively. These functions expect normal forms as arguments. Note that the term h(x) is not actually built by f . With the annotation [ ; 1] this is not possible. We have to build at least the term h(x), which must be passed to g before normalization. At this point we have to face some

*penalty compared to innermost rewriting, because term formation is relatively* expensive, especially for maximally shared terms.

## *5 Future Work*

*We mentioned that in many examples, the just-in-time strategy has a better* termination behaviour than innermost rewriting. We now generalize this to the following:

*Conjecture 5.1 Let R be a TRS with a full and in-time strategy annotation* strat . If t is strongly normalizing under the leftmost-innermost strategy, then strategy strat on t terminates.

*(If the rules are non-root-overlapping, \strongly" could be dropped). This* would be an important result, implicating that a rewrite implementation can make the transition from leftmost-innermost to just-in-time rewriting, without repercussions for the users. The improvement would be conservative, in the sense that all previous examples still terminate, and some more.

*We restricted attention to deterministic strategies. By dividing the an-* notations in groups, one could denote non-deterministic strategies. I.e. the innermost strategy (not just left-most innermost) is speci ed by an annota- tion like [f1; 2; 3g; f ; ; g]. In the proof machinery, rewrites must be replaced by sets of rewrites, and the deterministic sequences by non-deterministic tran- sition systems. In fact our proof is a bisimulation proof on sequences and this technique carries over to transition systems in a straightforward way [2](#_bookmark11) . How- ever, the straightforward implementation of annotation [f1; 2g] would either choose to normalize the rst argument completely, or the second which would not be memory-less. A memory-less strategy would allow alternations of steps in the rst and second argument.

*Another indication that the non-deterministic case is di erent is that the* following TRS (after Toyama), is a counter-example to our conjecture in the case of non-deterministic strategy annotations:

*: f (0; 1; x) 7! f (x; x; x) : 2 7! 0 : 2 7! 1*

*Any innermost reduction of f (0; 1; 2) terminates. But the non-deterministic* strategy indicated by f : [1; 2; ; 3] and 2 : [f ; g] allows an in nite reduction f (0; 1; 2) ! f (2; 2; 2) ! f (0; 2; 2) ! f (0; 1; 2) !

## *Acknowledgement*

*The strategy annotations of this paper date back to ideas of Jasper Kam-* perman and Pum Walters around 1996. At that time we tried a correctness

*2 History-sensitive non-deterministic strategies form a coalgebra: State ! P (Rewrite State), where State ' T erm M emory. The strategy is memory-less if State ' T erm; it is deterministic if the set on the right is a singleton.*

*proof along the lines of* [*[8,6,7],*](#_bookmark18) *which failed. After a recent implementation of* this strategy I tried a new proof of correctness. I thank Stefan Blom, Mark van den Brand, Jozef Hooman, Bas Luttik, Vincent van Oostrom and Hans Zantema for inspiring discussions and helpful hints. Finally, the referees were very helpful, by pointing out related work.

## *References*

*[1] F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University* *Press, 1998.*

*[2] P. Borovansky , C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with strategies in ELAN: a functional semantics. International Journal of* *Foundations of Computer Science, March 2001.*

*[3] M.G.J. van den Brand, H.A. de Jong, P. Klint, and P.A. Olivier. EÆcient annotated terms. Software { Practice and Experience, 30(3):259{291,* *2000.*

*[4] M.G.J. van den Brand, P. Klint, and P.A. Olivier. Compilation and memory management for ASF+SDF. In S. Jahnichen, editor, 8th Int. Conf. on Compiler construction CC'99, volume 1575 of LNCS, pages 198{213. Springer, 1999.*

*[5] M. Clavel, F. Dur an, S. Eker, P. Lincoln, N. Mart* *Oliet, J. Meseguer, and J.F.*

*Quesada. The Maude system. In P. Narendran and M. Rusinowitch, editors, Proc. of the 10th Int. Conf. on Rewriting Techniques and Applications (RTA- 99), volume 1631 of LNCS, pages 240{243. Springer, 1999.*

*[6] W. Fokkink, J. Kamperman, and Walters. P. Within ARM's reach: Compilation of left-linear rewrite systems via minimal rewrite systems. ACM Trans. on Programming Languages and Systems, 20(3):679{706,* *1998.*

*[7] W. Fokkink, J. Kamperman, and Walters. P. Lazy rewriting on eager machinery. ACM Trans. on Programming Languages and Systems, 22(1):45{ 86, January 2000.*

*[8] W. Fokkink and J.C. van de Pol. Simulation as a correct transformation of rewrite systems. In I. Pr vara and P. Ru zi cka, editors, 22th Int. Symp. on Mathematical Foundations of Computer Science (MFCS'97), LNCS 1295, pages 249{258, Bratislava, 1997. Springer.*

*[9] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, Software Engineering with OBJ: algebraic speci cation in action. Kluwer, 2000.*

*[10] J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In*

*M. Parigot and A. Voronkov, editors, Logic for Programming and Reasoning, LPAR2000, volume 1955 of LNAI, pages 161{178. Springer, 2000.*

*[11] J.W. Klop. Term rewriting systems. In D. Gabbay S. Abramski and*

*T. Maibaum, editors, Handbook of Logic in Computer Science, volume 2. Oxford University Press, 1992.*

*[12] S. Lucas. Computational properties of term rewriting with strategy annotations. In Proc. of Primeras Jornadas sobre Programacion y Lenguajes* *(PROLE'01), Universidad de Castilla-La Mancha, 2001 (to appear).*

*[13] S. Lucas. Termination of rewriting with strategy annotations. In*

*R. Nieuwenhuis and A. Voronkov, editors, Proc. of 8th Int. Conf. on Logic for Programming, Arti cial Intelligence and Reasoning, LPAR'01, volume 2250* *of LNAI. Springer, 2001 (to appear).*

*[14] S.P. Luttik and E. Visser. Speci cation of rewriting strategies. In M.P.A. Sellink, editor, Second International Conference on the Theory and Practice of Algebraic Speci cation (ASF+SDF'97), Electronic Workshops in Computing.* *Springer, 1997.*

*[15] T. Nagaya. Reduction Strategies for Term Rewriting Systems. PhD thesis, JAIST, Japan, 1999.*

*[16] A.T. Nakagawa, T. Sawada, and K. Futatsugi. CafeOBJ user's manual, version*

*1.3. Available via* [*http://www.ipa.go.jp/STC/CafeP/index-e.html.*](http://www.ipa.go.jp/STC/CafeP/index-e.html)

*[17] M. Nakamura and K. Ogata. The evaluation strategy for head normal form with and without on-demand ags. In K. Futatsugi, editor, The 3rd Int. W. on Rewriting Logic and its Applications (WRLA2000), volume 36 of Electronic Notes in Theoretical Computer Science. Elsevier, 2001.*

*[18] K. Ogata and K. Futatsugi. Implementation of term rewritings with the evaluation strategy. In H. Glaser, P. Hartel, and H. Kuchen, editors, 9th Int. Symp. on Programming Languages: Implmentations, Logics, and Programs, PLILP97, volume 1292 of LNCS, pages 225{239,* *1997.*

*[19] E. Visser. The Stratego tutorial and reference manual. Available via* [*http://www.stratego-language.org/.*](http://www.stratego-language.org/)

## *A Program Transformations*

*A.1 From norm to norm1* Take as de nition:

*norm1(t; p; L)= t[norm(tjp; L)]p*

*Then calculate:*

*norm1(t; p; []) = t[norm(tjp; [])]p*

*= t[tjp]p*

*= t*

*Let : l 7! r. If tjp = l, for some , then:* norm1(t; p; [ jL]) = t[norm(tjp; [ jL])]p

*= t[norm(r; strat (r))]p*

*= t[r]p[norm(t[r]pjp; strat (r))]p*

*= norm1 (t[r]p; p; strat (r))* If tjp 6= l, for any , then:

*norm1(t; p; [ jL]) = t[norm(tjp; [ jL])]p*

*= t[norm(tjp; L)]p*

*= norm1 (t; p; L)*

*Finally,*

*norm1(t; p; [ijL]) = t[norm(tjp; [ijL])]p*

*= t[norm(tjp[norm(tjpji; strat (tjpji))]i; L)]p*

*= t[norm(tjp[norm(tjp:i; strat (tjp:i))]i; L)]p*

*= fintroduce abbreviation Ag* t[norm(tjp[A]i; L)]p

*= t[A]p:i[norm(tjp[A]i; L)]p*

*= t[A]p:i[norm(t[A]p:ijp; L)]p*

*= norm1(t[A]p:i; p; L)*

*= norm1(t[norm(tjp:i; strat(tjp:i))]p:i; p; L)*

*= norm1(norm1 (t; p:i; strat(tjp:i)); p; L)*

*A.2 From norm1 to norm2* Take as de nition:

*Then calculate:*

*norm2 (t; []) = t*

*norm2(t; [(p; L)jS]) = norm2(norm1 (t; p; L); S)*

*norm2(t; [(p; [])jS]) = norm2 (norm1 (t; p; []); S)*

*= norm2 (t; S)*

*Let : l 7! r, if tjp = l for some , then:*

*norm2(t; [(p; [ jL])jS]) = norm2(norm1(t; p; [ jL]); S)*

*= norm2(norm1(t[r]p; p; strat(r)); S)*

*= norm2(t[r]p; [(p; strat(r))jS])*

*If tjp 6= l for any , then:*

*norm2(t; [(p; [ jL])jS]) = norm2(norm1(t; p; [ jL]); S)*

*= norm2(norm1(t; p; L); S)*

*= norm2(t; [(p; L)jS])*

*Finally,*

*norm2(t; [(p; [ijL])jS]) = norm2 (norm1(t; p; [ijL]); S)*

*= norm2 (norm1(norm1(t; p:i; strat (tjp:i)); p; L); S)*

*= norm2 (norm1(t; p:i; strat (tjp:i)); [(p; L)jS])*

*= norm2 (t; [(p:i; strat (tjp:i)); (p; L)jS])*

*A.3 From last(seq 2) to norm3* Take as a de nition:

*norm3(t; S) = last(seq 2(t; S))*

*We will use several times that if rewr2 (t; S) = rewr2 (t; R), then seq2 (t; S) =* seq2(t; R). Now calculate:

*norm3(t; []) = last(seq 2(t; []))*

*= frewr2 (t; []) = ?g* last(hti)

*= t*

*Next,*

*norm3(t; [(p; [])jS]) = last(seq 2 (t; [(p; [])jS]))*

*= frewr2 (t; [(p; [])jS]) = rewr2 (t; S)g* last(seq 2 (t; S))

*= norm3 (t; S)*

*Next, if : l 7! r and tjp = l, for some :* norm3(t; [(p; [ jL])jS])

*= last(seq 2(t; [(p; [ jL])jS]))*

*= frewr2 (t; [(p; [ jL])jS]) = (p; r; [(p; strat(r))jS])g*

*last(t :: seq2 (t[r]p; [(p; strat (r))jS]))*

*= last(seq 2(t[r]p; [(p; strat(r))jS]))*

*= norm3(t[r]p; [(p; strat(r))jS])* Otherwise, if tjp 6= l for any :

*norm3(t; [(p; [ jL])jS]) = last(seq 2(t; [(p; [ jL])jS]))*

*= frewr2 (t; [(p; [ jL])jS]) = rewr2 (t; [(p; L)jS])g* last(seq 2(t; [(p; L)jS]))

*= norm3(t; [(p; L)jS])*

*Finally,*

*norm3(t; [(p; [ijL])jS])*

*= last(seq 2(t; [(p; [ijL])jS]))*

*= frewr2 (t; [(p; [ijL])jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L)jS])g* last(seq 2(t; [(p:i; strat (tjp:i)); (p; L)jS]))

*= norm3(t; [(p:i; strat (tjp:i)); (p; L)jS])*

## *B Full Proofs of Lemma* [*3.4{3.7*](#_bookmark9)

*B.1 Proof of Lemma* [*3.4.(i).*](#_bookmark7)

*Let strat be in-time, let [(p; L)jS] be a well-formed stack, and assume that* rewr2 (t; [(p; L)jS]) = (q; s; R), where q 6 p. We now have to prove the follow- ing: rewr2 (t; [(p; L)jS]) = rewr2 (t; S). This is by induction on tjp and within that on L. We proceed by case distinction on L.

*L = []: In this case, rewr2 (t; [(p; [])jS]) = rewr2 (t; S) by de nition.*

*L = [l 7! rjL0 ]: If l = tj , then p = q, in contradiction with the assumption* q 6 p. So l 6= tjp for any . Then

*p*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p; L )jS])*

*0*

*= fInduction Hypothesis (L0 < L)g* rewr2 (t; S)

*L = [ijL0 ]: First note that if q p:i then q p.*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L )jS])*

*0*

*= fInduction Hypothesis (tjp:i < tjp)g* rewr2 (t; [(p; L )jS])

*0*

*= fInduction Hypothesis (L0 < L)g*

*rewr2 (t; S)*

*B.2 Proof of Lemma* [*3.4.(ii).*](#_bookmark7)

*Let strat be in-time, let [(p; L)jS] be a well-formed stack, and assume that* rewr2 (t; [(p; L)jS]) = (p; s; R). We have to prove: R = [(p; strat (s))jS]. This is by induction on L:

*L = []: Impossible, because p is not revisited from stack S (here well-* formedness of [(p; L)jS] is used).

*L = [ijL0 ]:*

*(p; s; R)= rewr2 (t; [(p; L)jS])*

*= rewr2 (t; [(p:i; strat (tjp:i)); (p; L0 )jS])*

*= fp 6 p:i, so use Lemma* [*3.4.(i)*](#_bookmark7)*g* rewr2 (t; [(p; L )jS])

*0*

*Hence by Induction Hypothesis (L0 < L), R = [(p; strat(s))jS].*

*L = [l 7! rjL0 ]: If tjp = l, for some , then R = [(p; strat(s))jS] by* de nition of rewr2 . Otherwise, tjp 6= l, for any , then

*(p; s; R) = rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p; L )jS])* Hence by Induction Hypothesis, R = [(p; strat (s))jS].

*0*

*B.3 Proof of Lemma* [*3.4.(iii).*](#_bookmark7)

*Let strat be in-time, let [(p; L)jS] be a well-formed stack, and assume that* rewr2 (t; [(p; L)jS]) = (q; s; R), with q 6 p. We now have to prove that rewr2 (t[s]q; [(p; L)jS]) = rewr2 (t[s]q; R).

*Starting with stack [(p; L)jS] and term t, rewr2 reduces in a number of* steps to (q; s; R). The proof proceeds by mimicking this reduction starting with the same stack in term t[s]q (see Figure [1).](#_bookmark6) The proof is by induction on the number of recursive calls of rewr2 (t; [(p; L)jS]) to (q; s; R). Distinguish cases for L.

*L = []. First, rewr2 (t; [(p; [])jS]) = rewr2 (t; S) and rewr2 (t[s]q; [(p; [])jS]) =*

*rewr2 (t[s]q; S). S =6 [], for then the result would be ?. By well-formedness*

*of S, p = p0 :j and S = [(p0 ; L0 )jS0 ] for some p0 ; L0 ; S0 . Note that q 6 p0 .* Hence by induction hypothesis, rewr2 (t[s]q; S) = rewr2 (t[s]q; R).

*L = [ijL0 ]. Then rewr2 (t; [(p:i; strat (tjp:i)); (p; L0 )jS]) =*

*rewr2 (t; [(p; [ijL0 ])jS]) = (q; s; R). Distinguish cases:*

*If p:i = q, then by Lemma* [*3.4.(ii),*](#_bookmark7) *R = [(p:i; strat (s)); (p; L0)jS].* rewr2 (t[s]q; [(p; [ijL0 ])jS])

*= rewr2 (t[s]q; [(p:i; strat (t[s]qjp:i)); (p; L0 )jS])*

*= rewr2 (t[s]q; [(p:i; strat (s)); (p; L0 )jS])*

*= rewr2 (t[s]q; R)*

*Otherwise, if p:i 6= q, then q 6 p:i. Note that the new stack is well-formed,* because i 62 L0 by the assumption that annotations have no duplicates. So the induction hypothesis can be used.

*rewr2 (t[s]q; [(p; [ijL0 ])jS])*

*= rewr2 (t[s]q; [(p:i; strat (t[s]qjp:i)); (p; L0 )jS])*

*= fhead(t[s]qjp:i) = head(tjp:i)g*

*rewr2 (t[s]q; [(p:i; strat (tjp:i)); (p; L0 )jS])*

*= fBy Induction Hypothesisg* rewr2 (t[s]q; R)

*L = [l 7! rjL0 ]. If tjp = l for some , then (q; s; R) = rewr2 (t; [(p; L)jS]) =* (p; r; [(p; strat (r))jS]), which contradicts q 6 p (in fact this case is dealt with in part 2).

*Hence tjp 6= l for any . We rst prove that t[s]qjp 6= l for any . This is* done by distinguishing two cases:

*If q >6*

*p, then t[s]qjp = tjp, so t[s]qjp 6= l for any .*

*If q > p, then q = p:i:p0 for some i; p0 . In this case i 2 L, because p:i will* not be revisited from S by the well-formedness of S. Because L is in-time, argument i is not needed by rule l 7! r, so by Lemma [2.1,](#_bookmark0) t[s]qjp 6= l, for any .

*Now (q; s; R) = rewr (t; [(p; L)jS]) = rewr (t; [(p; L0 )jS]). Similarly, we*

*2 2*

*have rewr (t[s] ; [(p; L)jS]) = rewr (t[s] ; [(p; L0 )jS]). By induction hypoth-* esis the latter equals rewr2 (t[s]q; R).

*2 q 2 q*

*B.4 Proof of Lemma* [*3.5.*](#_bookmark8)

*Let strat be in-time, and let rewr2 (t; [("; strat (t))]) = (q; s; R). We must prove:* rewr2 (t[s]q; R) = rewr2 (t[s]q; [("; strat(t[s]q))]).

*If q = ", then*

*rewr2 (t[s]"; R)= fby Lemma* [*3.4.(ii)*](#_bookmark7)*g*

*rewr2 (t[s]"; [("; strat(s))])*

*= rewr2 (t[s]"; [("; strat(t[s]"))])*

*Otherwise, q > ", so q 6 ". Then we have:* rewr2 (t[s]q; R)= fby Lemma [3.4.(iii)](#_bookmark7)g

*rewr2 (t[s]q; [("; strat(t))])*

*= fhead(t[s]q) = head(t)g*

*rewr2 (t[s]q; [("; strat(t[s]q))])*

*B.5 Proof of Lemma* [*3.6.*](#_bookmark10)

*We have to prove the following:*

1. *if rewr1 (tjp; L) = (q; s), then for some R, rewr2 (t; [(p; L)jS]) = (p:q; s; R).*
2. *if rewr1 (tjp; L) = ?, then rewr2 (t; [(p; L)jS]) = rewr2 (t; S).*

*The proof is by simultaneous induction, on tjp and within that on L. The* proof proceeds by case distinction on L:

*L = []:*

1. *rewr1 (tjp; L) = (q; s): Impossible*
2. *rewr1 (tjp; L) = ?: By de nition, rewr2 (t; [(p; [])jS]) = rewr2 (t; S).*

*L = [ijL0 ]:*

1. *rewr1 (tjp; L) = (q; s): Distinguish cases:*

*rewr (tj ; strat(tj )) = ?: Then rewr (tj ; L0 ) = rewr (tj ; L) = (q; s),* and

*1 p:i p:i 1 p 1 p*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L0)jS])*

*= fBy Induction Hypothesis (ii) (tjp:i < tjp)g*

*rewr2 (t; [(p; L )jS])*

*0*

*= fBy Induction Hypothesis (i) (L0 < L)g* (p:q; s; R) for some R

*rewr (tj ; strat(tj )) = (q0 ; s0 ): Then q = i:q0 and s0 = s.* rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L )jS])

*0*

*1 p:i p:i*

*= fBy Induction Hypothesis (i) (tjp:i < tjp)g*

*0 0*

*(p:i:q ;s ; R) for some R*

*= (p:q; s; R)*

1. *rewr (tj ; L) = ?: Then rewr (tj ; strat(tj )) = ? and rewr (tj ; L0 ) =*

*1 p 1 p:i p:i 1 p*

*?. Hence*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p:i; strat (tjp:i)); (p; L )jS])*

*0*

*= fBy Induction Hypothesis (ii) (tjp:i < tjp)g* rewr2 (t; [(p; L )jS])

*0*

*= fBy Induction Hypothesis (ii) (L0 < L)g*

*rewr2 (t; S)*

*L = [l 7! rjL0 ]:*

1. *rewr1 (tjp; L) = (q; s): Distinguish cases.*

*If tjp = l for some , then (q; s) = rewr1 (tjp; L) = ("; r) and* rewr2 (t; [(p; L)jS]) = (p; r; [(p; strat (r))jS])

*= (p:"; r; R) for some R*

*Otherwise, tj 6= l for any , so (q; s) = rewr (tj ; L) = rewr (tj ; L0 ),* and

*p 1 p 1 p*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p; L )jS])*

*0*

*= fBy Induction Hypothesis (i) (L0 < L)g* (p:q; s; R) for some R

1. *rewr1 (tjp; L) = ?: Then tjp 6= l for any . So ? = rewr1 (tjp; L) = rewr (tj ; L0 ), and*

*1 p*

*rewr2 (t; [(p; L)jS]) = rewr2 (t; [(p; L )jS])*

*0*

*= fBy Induction Hypothesis (ii) (L0 < L)g* rewr2 (t; S)

*B.6 Proof of Proposition* [*3.7.*](#_bookmark9)

*We must prove that seq2 (t; [("; strat(t))]) = seq 1(t). In order to present this* as a program transformation, we introduce the following speci cation as a de nition:

*seq 3(t) = seq 2(t; [("; strat(t))])*

*Now we calculate:*

*seq3(t)= seq2(t; [("; strat (t))])*

# *><8>*

*:>>*

*=*

*if rewr2 (t; [("; strat(t))]) = (q; s; R) for some q, s, R* then t :: seq 2(t[s]q; R)

*else hti*

*= fBy Lemma* [*3.5*](#_bookmark8)*g*

# *><8>*

*:>>*

# *><8>*

*:>>*

*=*

*if rewr2 (t; [("; strat(t))]) = (q; s; R) for some q, s, R*

*then t :: seq 2(t[s]q; [("; strat(t[s]q))]) else hti*

*if rewr2 (t; [("; strat(t))]) = (q; s; R) for some q, s, R* then t :: seq 3(t[s]q)

*else hti*

*= fBy Lemma* [*3.6*](#_bookmark10)*g*

# *><8>*

*:>>*

*if rewr1 (t) = (q; s) for some q, s then t :: seq 3(t[s]q)*

*else hti*

*This is exactly the de ning equation of seq 1.*