

Early reset and reference counting  
improve  
variable shunting in the WAM.

*Bart Demoen*

*Report CW 298, August 2000*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

**Early reset and reference counting  
improve  
variable shunting in the WAM.**

*Bart Demoen*

*Report CW298, August 2000*

Department of Computer Science, K.U.Leuven

**Abstract**

Sahlin and Carlsson claim that their variable shunting algorithm shunts all possible variable chains. We show two situations where this seems not the case. The algorithm can be adapted easily to take these situations into account. The key point is that marking and even early reset should be performed before variable shunting - and probably afterwards as well.

# Early reset and reference counting improve variable shunting in the WAM <sup>\*</sup>

Bart Demoen

Department of Computer Science  
Katholieke Universiteit Leuven  
B-3001 Heverlee, Belgium  
bmd@cs.kuleuven.ac.be

## Abstract

Sahlin and Carlsson claim that their variable shunting algorithm shunts all possible variable chains. We show two situations where this seems not the case. The algorithm can be adapted easily to take these situations into account. The key point is that marking and even early reset should be performed before variable shunting - and probably afterwards as well.

## 1 Introduction

Variable shunting is explained in detail in [5] with an algorithm (which we name SC-shunting in the future) and also in [4]. In [5] the authors claim that **all possible chains of variables are shunted by this algorithm**. This is of course not a formal statement and there is no proof of it. However, we will show two cases in which their algorithm can be improved: in the first case, it is shown that performing early reset (see for instance [1]) can result in more opportunities for shunting a variable. Early reset is tied intimately to marking and performed by most Prolog systems: performing marking with early reset before variable shunting is reasonable, but in section 3 we will indicate a potential pitfall. The second case shows that a (two valued) reference count for reachable cells can be used to improve variable shunting. Such reference counting is usually not performed in WAM, but systems that do not use mark bits in the heap itself can easily incorporate it.

It is not clear to what extent this is known and exploited in MALI [1] or other systems.

## 2 Case I: early reset can help variable shunting

Let A, B and C be three cells on the heap; let A point to B, B to C and C contain the atom a: see figure 1. Suppose furthermore that C is trailed - this is indicated by the shade in the figure: situation (1). SC-shunting will not be able to shunt anything. However, assume that early reset is performed first and that cell C is subject to early reset <sup>1</sup>, then early reset will arrive at situation (2) in figure 1: C is no longer on the trail and an undef. Finally SC-shunting will arrive at situation (3) which is better than what can be achieved without early resetting before shunting.

---

<sup>\*</sup>Another case of ill-understood usefulness logic ?

<sup>1</sup>it is a mind teaser to construct this with Prolog code ... you also need a compiler that does choicepoint trimming - see also [2]

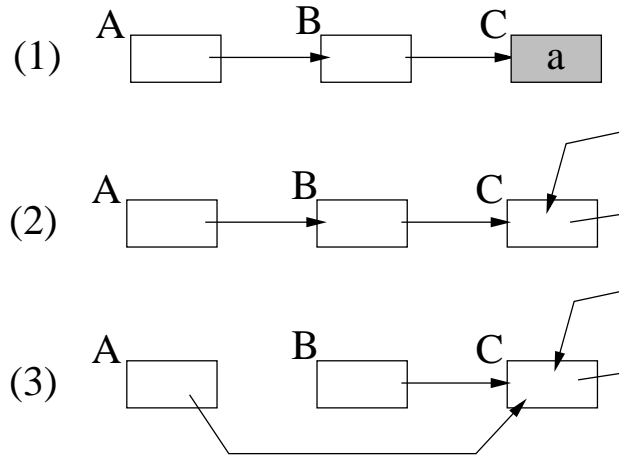


Figure 1: Shunting improved by early resetting first

### 3 Marking before variable shunting

When using sliding compaction, the number of marked cells is counted, because the sliding phase needs to know the number of marked cells. Since the marking has taken place before the shunting, let's name this number  $M_{before}$ . One can do a marking after the shunting as well; this gives us the number  $M_{after}$ . It is clear that it is possible that  $M_{after} < M_{before}$ . Which means that for a more optimal gc, one must mark again after variable shunting. This is not necessary when using a copying compaction algorithm, because the exact number of live cells is not important - as long as a decent upper bound can be given.

### 4 Case II: reference counts can help variable shunting

One characteristic of SC-shunting is that no cell ever becomes an undef in the process: we call this *undefining a cell*. One reason is that doing so potentially makes reference chains longer; an attempt of undefining some cells during shunting was made while implementing the BinProlog garbage collectors ([2]), but only for what was named *directly reachable cells*. However the method could result in longer reference chains. Let us first show that undefining is in general wrong: in the left part of figure 2 there are variables A and B both pointing at C which is an undef. Replacing A and B by an undef as in the right part of the picture, loses the sharing between A,B and C and is wrong.

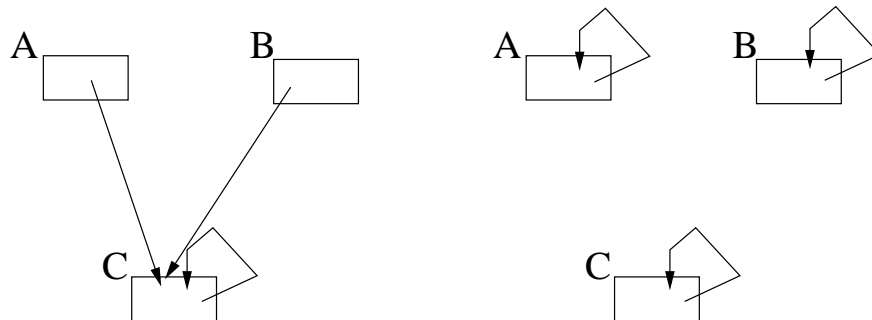


Figure 2: Bad shunting

The problem is intimately related to sharing and reference counts can capture this sharing. Before introducing the use of reference counts, let us first describe a situation in which SC-shunting can't do much: suppose that three variables A, B and C are in the same segment, and A is bound to B and B to C, so that the situation in the upper half of figure 3 is produced (for ease of understanding, think of A as an element of a list: there is no reference to A from the control stack). Assume also that none of the variables is trailed. Then SC-shunting will end up with the situation in the lower half of figure 3.

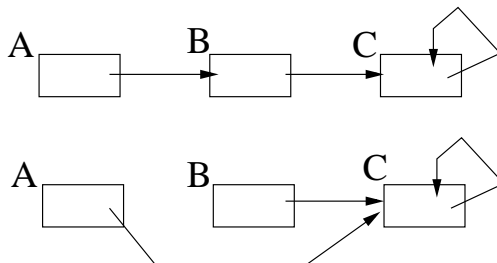


Figure 3: SC-shunting

If reference counts are taken into account, one can do better. Suppose that during the marking phase we have also kept a two valued reference count: its value can just be *one* or *more*. Assume still that no variable is trailed. The better shunting is shown in figure 4.

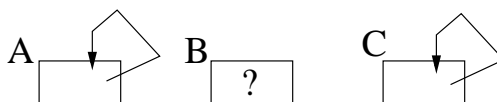


Figure 4: Improved shunting with reference counts

The (real) reference counts of B and C have dropped to zero and they are unreachable now - that's why B contains a ?: its value is not important.

The general principle is: if a cell references an undef which has reference count *one* then that cell can be undefed - modulo the usual considerations for the trail.

But even if B is trailed, we can do better: in the picture, we'll indicate that a cell is trailed by shading it. Consider the situation (1) in figure 5. Shunting B leads to situation (2), in which B is trailed but contains already an undef. This means that the reference to B from the trail can be deleted, so that we arrive at situation (3) and finally after one more step at situation (4).

This example shows that shunting doesn't need to stop on trailed cells (or in the jargon of [5]: a cell with an older binding). It also shows that during shunting, segment boundaries can be crossed. Finally, it is important to realize that the references on the trail do not contribute to the reference count of a cell. Indeed, in general one can say that the trail does not contribute positively to usefulness.

It is clear that trying to remove the trail reference to B from the trail, is a potentially costly operation. But keep in mind that after shunting, a new marking phase is beneficial: at that moment B will disappear from the trail.

A final example shows that the reference count can also improve shunting when the end of the chain is not an undef. Figure 6 shows a situation in which A is bound to B, B to C (with B trailed) and C to the atom a. All cells have reference count *one*. Situation (2) is obtained by normal SC-shunting. Situation (3) is obtained correctly because A and B have reference count *one*

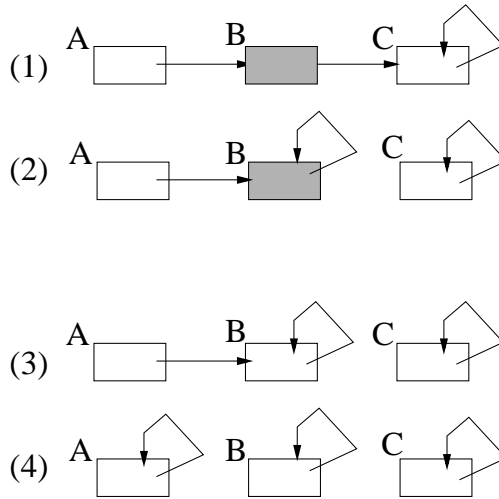


Figure 5: Improved shunting with reference counts in the presence of trailing

in situation (2): B now because unreachable and can be removed from the trail. On the other hand, A must be put in the trail at the place where B was, so that on backtracking, A can be reset to undef - unless the trail entry for B appears in later than the segment in which A was created, in which case A need not appear on the trail.

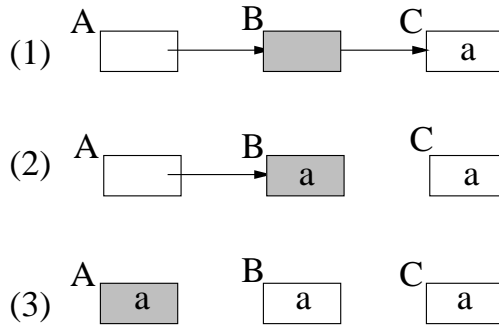


Figure 6: Improved shunting with reference counts in the presence of trailing and bound cells

Again, removing B from the trail should be done in a marking phase after the shunting. Replacing the B entry with A, is more complicated: if there are enough bits (either tag bits in the cells, or in the mark array) one could set such a bit and make B point to A at the moment of the shunting. Subsequent marking can then - during the treatment of the trail - test that bit and take appropriate action.

## 5 Conclusion

We claim by no means that with the two extensions described above, all possible chains are shunted. Neither that it is worthwhile to implement the extensions in a real system. However, performing marking (and early rest) before variable shunting is easy and there are other uses for the reference count (see forthcoming report) so it would not be unreasonable to do so.

This work originated from our work on specifying garbage collection in logic [3]. This report

indicates that the usefulness logic of the WAM - although perhaps abstractly well understood - needs better understanding on the practical level as well.

## References

- [1] Y. Bekkers, O. Ridoux, and L. Ungaro. Dynamic memory management for sequential logic programming languages. In Y. Bekkers and J. Cohen, editors, *Proceedings of IWMM'92: International Workshop on Memory Management*, number 637 in LNCS, pages 82–102, St. Malo, France, Sept. 1992. Springer-Verlag.
- [2] B. Demoen, G. Engels, and P. Tarau. Segment preserving copying garbage collection for WAM based Prolog. In *Proceedings of the 1996 ACM Symposium on Applied Computing*, pages 380–386, Philadelphia, Feb. 1996. ACM Press.
- [3] B. Demoen. Prolog and abduction 4 writing garbage collectors Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-00-6-1, URL : <http://www.cs.man.ac.uk/cstechrep/titles00.html> Proceedings of LOPSTR 2000 - Tenth International Workshop on Logic-based Program Synthesis and Transformation; July, 2000,
- [4] S. Le Huitouze. *A new datastructure for implementing extensions to Prolog*. Proceedings of the International Workshop on Programming Language Implementation and Logic Programming (PLILP'90), LNCS 456, pp. 136-150, 1990
- [5] D. Sahlin and M. Carlsson. *Variable shunting for the WAM* SICS research report R91:07