Addendum to “Optimising the Linda in primitive: Understanding tuple-space run-times" presented at SAC’2000

Antony Rowstron
Microsoft Research
1 Guildhall Street
Cambridge, CB2 3NH, UK
antr@microsoft.com

ABSTRACT

Subsequent work on the formal proof of the optimisation described in the paper has shown a particular case when the optimisation allows behaviour that is not possible using the traditional Linda semantics[2]. The case that causes this is described, and a simplification of the method is presented which provides nearly the same degree of optimisation without altering the semantics.
The original optimisation algorithm does not work when deadlock is explicitly employed in the program (by causing a tuple space access primitive to block (forever) a thread of computation). Although unlikely, in an open system this makes the optimisation unacceptable. However, the method as presented in the paper could be used as an optimisation run-time flag in closed implementations, such as the SCA C-Linda[1].
The simplified optimisation algorithm has no implementation impact. It can be used in open implementations and supports optimisation of the motivating examples given in the original paper.

1. INTRODUCTION

In order to demonstrate the problem with the current optimisation algorithm consider the following two agents performing the operations:

	Agent A		Agent B
A_1	in(a)	B_1	in(b)
A_2	rd(b)	B_2	rd(a)

Let us assume that these two agents are accessing the same tuple space, and that no other agents are able to access the tuple space, and that the tuple space contains two tuples, a and b. The case graph for these agents is shown in Figure 1. As can be seen, there is no case where both Agent A and B complete, either one or both block waiting for a matching tuple (which will never arrive). However, when the optimisation is being used, both agents will terminate having performed all the operations.

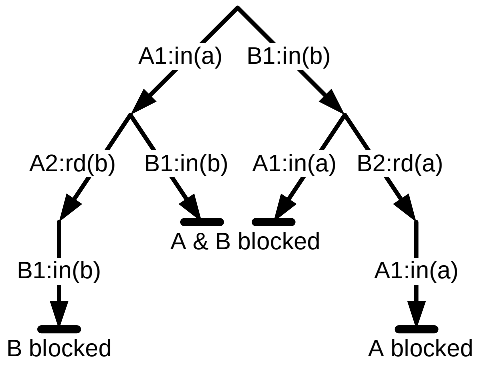

Figure 1: Case graph for agents A and B

The problem is that the programmer has explicitly expected one of the agents not to terminate. In a closed system, such as the SCA C-Linda, it is possible for a programmer to know whether they are expecting tuple space accesses to block forever and to switch the optimisation on accordingly. This may seem unlikely, however, it is possible, and some years ago, we worked on the parallel implementation of a stable marriages algorithm that explicitly used deadlock. However, for open systems it is not possible to know whether people are explicitly using deadlock or not, so the method as described in the paper is not acceptable for an open system. However, it is possible to simplify the algorithm slightly.

2. SIMPLIFIED ALGORITHM

This optimisation algorithm presented here is a simplified version of the optimisation algorithm presented in Section 3.2 in the original paper which is the general version of the algorithm covering Linda extensions. The differences between the two algorithms are highlighted in *italics* in this version.
When a tuple is to be used as a result or part of a result for a primitive and the tuple has been the result or part of a result for a destructive primitive then the tuple can still be used as a result to another primitive providing the following are all true:
i. The primitive being performed is not destructive.

ii. The primitive is not being performed by the same agent that performed the destructive primitive.
iii. The agent that performed the destructive primitive has not *performed any tuple space access since the destructive primitive.*
iv. The primitive being performed blocks the agents thread of execution until a result is returned, where a result is either a tuple or an indication of completion of some movement of tuples[1].
The system must subsequently discard this tuple when:
i. The agent that performed the primitive that removed the tuple *performs any operation on a tuple space.*
ii. The agent that performed the primitive that removed the tuple terminates.
iii. The current non-destructive primitive does not block the user thread of execution until a matched tuple(s) is found or operation on a set of tuples is complete, and is forced to use the tuple to provide the results correctly.
This should be transparent to the programmer, and the semantics of the primitives should not appear to be different from the traditional semantics of the Linda primitives.

3. DOES IT STILL REDUCE BLOCKING OF RD PRIMITIVES?

The motivation for the work was to reduce the number of `rd` primitives that have to block. In particular, when tuples are being used as shared counters as the example in Figure 1 of the main paper demonstrates. In this case, the simplified optimisation algorithm still allows the reader agents not to block, which is the desired behaviour. Therefore, the simplified optimisation algorithm still provides a performance increase for a run-time system using it. However, with the original algorithm, an `in` could be followed by an arbitrary number of `rd` primitives and the tuple consumed by the `in` would remain partially visible. However, we the simplified optimisation algorithm this is no longer the case.

4. IMPACT ON IMPLEMENTATION

The implementation has to be only slightly altered to support the simplified optimisation algorithm, instead of just incrementing the primitive count when an agent performs an `out` operation, the counter is incremented after every tuple space access operation. The overhead added to an `in` and `rd` of incrementing the primitive counter is not noticeable (the results shown for a `out` in the paper demonstrate the cost of incrementing the primitive count is not significant). No other extra costs are added to an `in`, `rd` or `out` by using the simplified algorithm. The experimental results as presented in Table 1 in the original paper still hold.

5. REFERENCES

[1] S. C. Associates. *Linda: User's guide and reference manual.* Scientific Computing Associates, 1995.

[2] R. D. Niccola. Private communication, 1999.

[1]Primitives such as `rdp` fail this rule, because they do not block the thread of execution – it returns false if a tuple is not available. However, a primitive like `copy-collect` passes the rule because it copies tuples and then returns a counter.