University of Bristol

DEPARTMENT OF COMPUTER SCIENCE

A Note on the Proof of Reduction of Biconnected Digraphs to Normal Forms

H Wilson J Bradley

September 1999 | CSTR-99-009

A Note on the Proof of Reduction of Biconnected Digraphs to Normal Forms

Helen Wilson* Jeremy Bradley†

Abstract

We present a proof method for reducing biconnected digraphs to a normal form and demonstrate its convergence for a general biconnected digraph.

1 Introduction

We have a requirement to show that biconnected digraphs (directed graphs in which every node is reachable from any other node) can be reduced to one of the normal forms shown in figure 1.

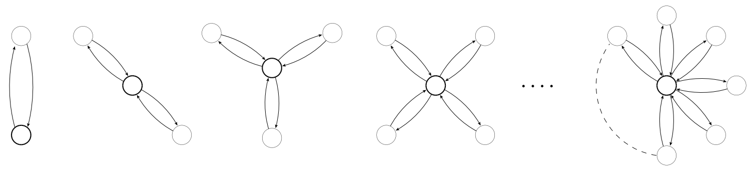

Fig. 1. The set of normal forms for a state with finite branching degree.

The reduction method presented in this report is applied around a chosen node in the digraph so that the out-edges from that node are preserved. Therefore the out-degree of the chosen node dictates which normal form will result from the method and the node itself becomes the highlighted node shown in figure 1.

The method is used in [1] and [2] to demonstrate that the steady-state distribution of a basic stochastic process with no concurrency can always be found. (For further details of the application area this was needed for, see either [1] or [2].)

*Department of Chemical Engineering, University of Colorado, Campus Box 424, Boulder, CO 80309-0424, USA.Email: Helen.Wilson@colorado.edu, Tel: +1 303 492-0475, Fax: +1 303 492-4341

†Department of Computer Science, University of Bristol, Woodland Road, Bristol BS8 1UB, UK. Email: Jeremy.Bradley@bristol.ac.uk, Tel: +44 117 903-1112, Fax: +44 117 954-5208

2 Reduction Rules

Five reduction rules are constructed and shown in figure 2.

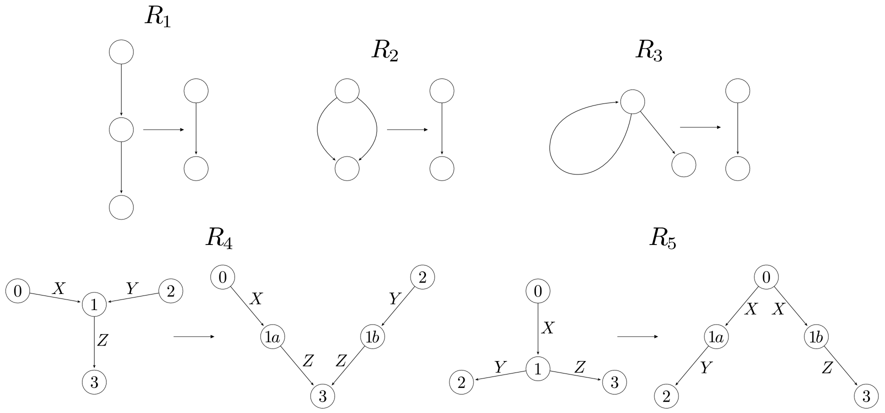

Fig. 2. The digraph reduction rules.

R_1 A node with precisely one in-edge and one out-edge may be removed, and the two edges replaced by a single edge.

R_2 When there are two edges with the same start and end vertices, one of them may by removed.

R_3 An edge which begins and ends at the same vertex may be removed.

R_4 A node with more than one in-edge may be replaced by a pair of nodes. Each new node has out-edges identical to the original node. The in-edges of the original node are divided between the two new nodes, each in-edge going to precisely one of the new nodes. After the rule has been applied, one of the two new nodes has exactly one in-edge. (The illustration in figure 2 shows the special case in which the original node has precisely two in-edges.)

R_5 A node with more than one out-edge may be replaced by two new nodes in a process exactly analogous to R_4.

3 Description of Reduction Algorithm

In the argument below, a *circuit* is defined to be a cyclic path of a digraph in which no node is ever traversed twice through the *same* pair of in- and out-edges. There is a finite number of such circuits in a strongly connected digraph. We also define the *parents* of a given vertex, v, to be the set of adjacent vertices which are connected to v via their out-edges. The *children* of v are the set of vertices which have v as a parent.

Let G be a strongly connected or biconnected digraph:

- It is possible to get to any vertex from any other vertex.
- For a chosen vertex v in G, v is an element of each member of the set C of circuits which contain v and not an element of any member of the set C' of circuits which do not contain v.
- $G = C \cup C'$
- We require to remove C' by aggregation and reduce C to one of the normal forms of figure 1, where the out-degree (number of out-edges) of v is preserved.

It can be shown (section 4) that R_2 and R_3 reduce the number of circuits in a digraph, whereas R_1, R_4 and R_5 preserve the number of circuits. By applying the reduction rules to a general digraph, we can only reduce or maintain the number of circuits. In the algorithm given below, each iteration uses at least one instance of R_2 or R_3, so the number of circuits decreases monotonically.

We describe the following digraph reduction method, to remove C':

1. Select a vertex v.
2. Remove all immediately reducible circuits with R_2, R_3.
3. Select the circuit $c \in C'$ of shortest length (c does not contain v).
4. Select a vertex $x \in c$ and label it as the root of the circuit. Let $u := x$:
   (a) Go to u', the child of u in circuit c; if u' has more than one parent then split using R_4 into a vertex with u as its only parent and a vertex with the remaining parents of u'.
   (b) Let $u := u'$, and $u' :=$ the child of vertex u' in c, whose parents we may just have increased. Repeat (4a) until all vertices in c, except x, have been processed as u' (as in figure 3).
5. Essentially we now perform 4 in reverse using R5 (figure 4). As before, let $u := x$:
   (a) Go to u', the parent of u in circuit c; if u' has more than one child then split using R_5 into a vertex with u as its only child and a vertex with the remaining children of u'.
   (b) Let $u := u'$, whose children we may just have increased. Repeat (5a) until all vertices in c, except x, have been processed as u'.
6. Apply R_1 to all the vertices of circuit c except x, until x is the only vertex left in c, with a single edge to itself.
7. Apply R_3 to remove c from C'.
8. Repeat steps (3)–(7), until C' is empty.

Now we need to restructure G (which now consists entirely of C, i.e. circuits containing v), to look like one of the normal forms of figure 1. Our goal is to maintain the out-edges from v unaggregated.

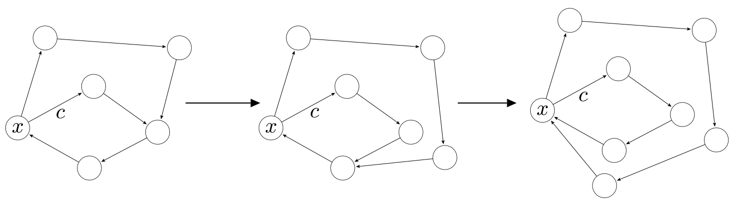

Fig. 3. Rewriting the circuit c to have no vertices with multiple parents except x.

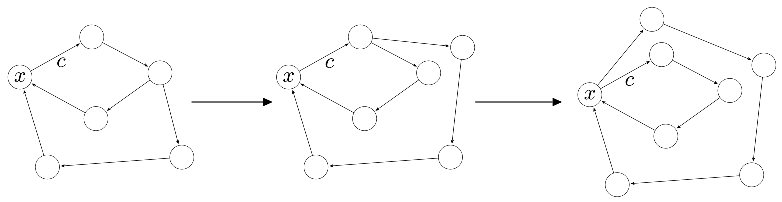

Fig. 4. Rewriting the circuit c to have no vertices with multiple children except x.

1. Select the shortest circuit of length > 2 through v and reapply steps (4)–(6) with $x := v$.
2. Apply R_1 to all vertices of this circuit except v and child of v.
3. Apply R_2 to the remaining in-edges to v.
4. Repeat for each circuit through v until G is in normal form.

4 Proof of Convergence of the Method

As noted in section 3 above, there is a finite number of circuits in any finite strongly connected digraph.

Theorem 1 *R_2 and R_3 reduce the number of circuits in the graph by at least one.*

This fact is self-evident: R_3 removes a circuit of length one and, for R_2, since at least two circuits originally passed through the pair of nodes given, one through each edge (by the strongly connected property), at least one of these circuits will be destroyed.

Theorem 2 *None of R_1, R_4 and R_5 can increase the number of circuits in the graph*

- For R_1 this fact is self-evident. The number of original circuits which pass through the node to be removed is conserved as the number of circuits which pass through the remaining edge after application of the rule.

- R_4 and R_5 are exactly equivalent in graph-theoretic terms, applying as they do to edges acting in opposite directions.
- We deduce that the above statement requires proof only for the case of R_4.

Proof. Let us label our chosen node v. After application of R_4, it becomes the pair of nodes $\{v_1, v_2\}$. We will consider the circuits which pass through v_1 and v_2, and show that the total number of circuits passing through $\{v_1, v_2\}$ is equal to the original number of circuits passing through the node v.

In either graph, we consider circuits which pass through either v or $\{v_1, v_2\}$. We can label each pass through these nodes by its input and output edges. (The part of the circuit which is not involved with these nodes will be unchanged under use of R_4.) Now those passes which chose node v_1 exactly correspond to those passes in the original graph which use the parent of v_1, and those which used v_2 to all other passes. Each such pass has an out-edge whose destination node is unchanged in the transformation. Therefore we have constructed a bijection between circuits in the original graph and those in the graph after application of R_4. Thus R_4 preserves exactly the number of circuits in a graph.

Acknowledgements

The authors would like to thank Ian Holyer for his advice and valuable discussions about this paper.

References

[1] Bradley, J. T. *Towards Reliable Modelling with Stochastic Process Algebras.* PhD thesis, Department of Computer Science, University of Bristol, Bristol BS8 1UB, UK, October 1999.

[2] Bradley, J. T., and Davies, N. J. An aggregation technique for analysing some generally distributed stochastic processes. CSTR Technical Report CSTR–99–003, Department of Computer Science, University of Bristol, Bristol BS8 1UB, UK, March 1999.

[3] Gibbons, A. *Algorithmic Graph Theory.* Cambridge University Press, 1985.