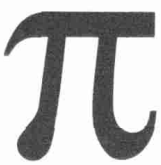
Department of Philosophy - Utrecht University

Proving a graph well founded using resolution

M. Bezem
J.F. Groote



Logic Group
Preprint Series
No. 113
May 1994

Utrecht Research
Institute for
Philosophy

ISBN 90-393-0952-3
ISSN 0929-0710
Dr. A. Visser, Editor

Proving a Graph Well Founded using Resolution (a case study in automated verification)

Marc Bezem
Jan Friso Groote
Department of Philosophy, University of Utrecht
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
Email: Marc.Bezem@phil.ruu.nl, JanFriso.Groote@phil.ruu.nl

Abstract

Using the resolution theorem prover OTTER we prove that a certain directed graph has no infinite path. We argue that this technique complements the well known combinatorial algorithms in cases in which the graph has a symbolic definition and a very large (or infinite) vertex set. The graph arose in the verification of a communication protocol.

Key Words & Phrases: process resolution, graph, verification of distributed systems.
1991 Mathematics Subject Classification: 68Q20, 68Q60, 68R10.
1987 CR Categories: F.3.1, F.4.1, I.2.3.
Note: The authors are partly supported by the Netherlands Computer Science Research Foundation (SION) with financial support of the Netherlands Organization for Scientific Research (NWO).

1 Introduction

Consider the trivial graph[1] on the natural numbers defined by the following edge set E:

$$E = \{(n, n+1) \in \mathbb{N} \times \mathbb{N} \mid even(n)\}.$$

Obviously, E is well founded[2], i.e. has no infinite paths, as all paths have length 0 or 1. Suppose we want to establish this fact by automatic means. Combinatorial algorithms (i.e. algorithms working on an explicit graph representation) do not immediately apply since the vertex set is infinite. Even if we would restrict ourselves to, say, a finite initial segment of the natural numbers, then combinatorial algorithms (for example, acyclicity tests) do not supply an attractive solution if the initial segment of the natural numbers is large. In both cases it seems better to exploit the symbolic nature of the definition of the graph and to look for a symbolic proof of the well-foundedness of the graph.

Let (V, E) be a graph with vertex set V and edge set $E \subseteq V \times V$. To be able to reason about well-foundedness we introduce a predicate WF on V. The intuitive meaning of $WF(x)$ is that there exists no infinite path in the graph starting with x. The predicate WF has an

[1] As all graphs in this paper will be directed, we simply say 'graph' instead of 'directed graph'.
[2] The ordering decreases in the direction of the edges.

inductive definition: it is the smallest predicate (in the sense of set inclusion) satisfying, for every $x \in V$, that $WF(x)$ holds whenever $WF(y)$ holds for all $y \in V$ such that there exists an edge from x to y in E. Note that the predicate WF is well defined since the above condition is closed under arbitrary intersection. Now the graph is well founded if $WF(x)$ holds for any $x \in V$.

Reconsider the trivial graph above. In this case the WF predicate satisfies:

$$\forall n \in \mathbb{N}\ ((even(n) \rightarrow WF(n+1)) \rightarrow WF(n)). \tag{1}$$

The well-foundedness of the graph is most easily proved by contradiction. Assume $\neg WF(n)$ for some $n \in \mathbb{N}$. Then $\neg(even(n) \rightarrow WF(n+1))$, so $even(n) \wedge \neg WF(n+1)$. Similarly, $\neg WF(n+1)$ implies $even(n+1) \wedge \neg WF(n+2)$. Now $even(n) \wedge even(n+1)$ leads to the desired contradiction.

Observe that the above argument is completely within the realm of first order logic: given some basic information on the predicate *even* on the natural numbers, we have shown $\forall n \in \mathbb{N}\ WF(n)$ for every predicate WF satisfying equation (1), so in particular for the smallest of all such predicates WF. This shows that the inductive definition of WF does not always prevent the use of first order theorem proving techniques.

There are several ways to automate reasoning in first order logic. In this paper we chose for resolution [7] and the resolution based theorem prover OTTER [5, 6, 8]. For the trivial graph above, OTTER establishes well-foundedness instantaneously in four reasoning steps. However, theorem provers have difficulties with solving complicated reasoning problems. In order to investigate whether theorem provers are effective in proving well-foundedness of non-trivial graphs, we have undertaken an experiment on which we will report in this paper.

The graph that we will prove well founded arose in a verification of a bidirectional one-bit sliding window protocol [2] in the setting of process algebra, extended with data. It took both authors a week to prove this graph well-founded. The protocol has been specified and verified in μCRL [1, 3, 4]. The state space of the protocol, which will be the vertex set of the graph, is essentially finite and has about 155.000 states (to be multiplied by $|D|^6$, where $|D|$ is the size of data set D).

It turned out to be in no way trivial to have OTTER generate the proof by itself and some problem specific methods were needed to have it done at all. This is mainly a problem of scale. We believe that the symbolic approach is in fact quite general and we encourage readers to try and improve on our results.

It is not useful to present the protocol in full detail here. Instead, we specify an example process in μCRL which signals a if an internal counter is odd, and performs an internal step τ if the internal counter is even. In both cases it increments its internal counter with one, and then repeats the above procedure.

$$\mathbf{proc}\ \ X(n{:}\mathbb{N}) = \tau\, X(n+1) \lhd even(n) \rhd a\, X(n+1). \tag{2}$$

Here, $\_ \lhd \_ \rhd \_$ must be read as *then_if_else_*. We say that such an equation is convergent if no infinite sequence of internal steps can be performed in any state $X(n)$. Convergent equations have unique solutions in many natural process algebras. They are used as a definition mechanism in such process algebras. The equation (2) is convergent since the graph presented in the first sentence of the introduction is well founded.

Apart from the above process theoretic motivation, the well-foundedness of graphs is obviously related to the termination of recursive programs, term rewriting systems, etcetera.

2 Some theory

The sentence 'This shows that the inductive definition of WF does not always prevent the use of first order theorem proving techniques.' from the introduction requires some explanation in order to prevent misunderstandings. Let T_E be a theory defining the graph E. In the simple example of the introduction, T_E could be Peano Arithmetic together with the formula $E(x,y) \leftrightarrow (even(x) \wedge y = x+1)$. Let $PROG_E(\phi)$, with ϕ an arbitrary unary predicate, be defined by:

$$PROG_E(\phi) \equiv \forall x(\forall y(E(x,y) \rightarrow \phi(y)) \rightarrow \phi(x)).$$

With E as above, $PROG_E(WF)$ becomes $\forall x(\forall y(even(x) \wedge y = x+1 \rightarrow WF(y)) \rightarrow WF(x))$, which is of course equivalent to (1). By the above definition, $PROG_E(\phi)$ expresses that ϕ is *progressive* with respect to E, i.e. that $\phi(x)$ holds whenever $\phi(y)$ holds for all y such that $E(x,y)$. We will simply call such ϕ progressive without the qualification 'with respect to E' if it is clear from the context which E is meant. Furthermore, we abbreviate $\forall x(\psi(x) \rightarrow \phi(x))$ by $\psi \subseteq \phi$. Now WF can be axiomatized by the following (second order) formula:

$$PROG_E(WF) \wedge \forall \phi(PROG_E(\phi) \rightarrow WF \subseteq \phi).$$

This formula expresses that WF is the smallest progressive predicate. The following lemma states that we can sometimes omit the second order part of the above formula, which reads $\forall \phi(PROG_E(\phi) \rightarrow WF \subseteq \phi)$.

Lemma 2.1. *Let T_E be a theory. Consider the following model theoretic statements:*
(a) $T_E \wedge PROG_E(WF) \models \forall x\, WF(x)$;
(b) $T_E \wedge PROG_E(WF) \wedge \forall \phi(PROG_E(\phi) \rightarrow WF \subseteq \phi) \models \forall x\, WF(x)$.
We have that (a) implies (b) and, if the language of T_E does not contain the predicate symbol WF, also (b) implies (a).

Proof. The first implication is trivial. For the second implication, assume (b) and let WF not be contained in the language of T_E. Let M be a model of $T_E \wedge PROG_E(WF)$. We have to show that $M \models \forall x\, WF(x)$. We have that $[\![WF]\!]_M$ is a progressive predicate. So the set of progressive subsets of the domain of M is non-empty. Moreover, it is easily read off from the definition of $PROG_E(\phi)$ that the set of progressive subsets of the domain of M is closed under arbitrary intersection. It follows that there exists a smallest progressive subset of the domain of M. Let M' be the interpretation, which is obtained from M by taking for $[\![WF]\!]_{M'}$ the smallest progressive subset of the domain of M. Since WF does not occur in T_E we have that M' is a model of T_E, and by our choice of $[\![WF]\!]_{M'}$ also of $PROG_E(WF) \wedge \forall \phi(PROG_E(\phi) \rightarrow WF \subseteq \phi)$. By (b) it follows that M' is a model of $\forall x\, WF(x)$. Since $[\![WF]\!]_{M'} \subseteq [\![WF]\!]_M$, we have that $\forall x\, WF(x)$ is also true in M. □

The previous lemma ensures soundness of omitting the second order part of the definition of $PROG_E$. If T_E is a first order theory, then statement (a) of the lemma forms a first order

theorem proving problem. Completeness has been restricted in the lemma to cases in which WF does not occur in the language of T_E. Let us examine how relevant this restriction is. The following lemma is due to Albert Visser.

Lemma 2.2. *Let T_E be a first order theory whose language does not contain WF. Then the following are equivalent:*
(a) $T_E \wedge PROG_E(WF) \models \forall x\, WF(x)$;
(b) there is a uniform bound such that in every model of T_E every path of E is shorter than this bound.

Proof. It is obvious that (a) follows from (b). For the converse, assume not (b). Introduce countably many new constant symbols $c_0, c_1, \ldots$. Let, for every $n \in \mathbb{N}$, T_n be the theory extending T_E with axioms $E(c_i, c_{i+1})$ for all $i < n$. Since not (b), every T_n has a model. Hence by the Compactness Theorem $\bigcup_{n\in\mathbb{N}} T_n$ has a model, and the interpretations of the constants $c_0, c_1, \ldots$ establish an infinite path in E in this model. It follows that (a) does not hold. □

At first sight, this lemma seems to show the limited applicability of the proposed method: first order theorem proving techniques can only prove a graph well founded when it satisfies condition (b). There are three counterarguments against this observation. First, in the case of finite graphs the existence of a uniform bound on the length of paths is equivalent to being well founded (and to being acyclic). Second, there are many practical cases of infinite graphs which are well founded since there exists a uniform bound on the length of paths. Third, there is nothing against extending T_E with true axioms in which WF occurs. In that case we rely on soundness, Lemma 2.1(a).

We finish this section with an example in which T_E should be extended with an axiom about WF. Let T_E be Peano Arithmetic plus the formula $E(x, y) \leftrightarrow x > y$. It seems obvious that E is well founded, but how does one prove this? Obviously, $PROG_E(WF)$ implies $WF(0)$ and $\forall x(WF(x) \rightarrow WF(x+1))$. It is tempting to conclude that $\forall x\, WF(x)$ holds by induction. However, WF does not occur in the language of Peano Arithmetic, so formulas in which WF occurs are *not instances of the induction schema.*[3] Adding the second order part of the definition of WF, $\forall\phi(PROG_E(\phi) \rightarrow WF \subseteq \phi)$, does not help. The only thing which helps is postulating induction for WF, i.e. $(WF(0) \wedge \forall x(WF(x) \rightarrow WF(x+1)) \rightarrow \forall x\, WF(x)$.

3 The graph in question

We proceed with presenting our case study. Point of departure is the recursive linear form X of the one-bit sliding window protocol from [2, Section 6]. We only provide those details of X that are necessary for a proper understanding of this paper. The process X has 18 parameters, as shown by the following call:

$$X(ready_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, ready_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2).$$

[3] In certain non-standard models of Peano Arithmetic induction with respect to formulas in which WF occurs does not hold!

Here $ready_1$, rec_1, sts_1, $ready_2$, rec_2, sts_2 are of type boolean, so either t or f. The parameters p_1, q_1, p_2, q_2 are bits, so either 0 or 1. The parameters d_1, e_1, d_2, e_2 are of some arbitrary (not necessarily finite) data set D. The parameters st_1, st_2 are status parameters and are either *read* or *choice* or *del*. Finally, the parameters f_1, f_2 are triples (frames) $\langle d, p, q\rangle$, where d is a datum and p, q are bits.

Below we give an exhaustive list of recursive calls of process X from via a τ-step [2, Section 6], followed by the condition under which the call may take place. We use the following notational convention: in a recursive call, we only make explicit those parameters that change with respect to the original call. For example, f/rec_1 means that f is the new value for parameter rec_1, $inv(p_1)/p_1$ means that parameter p_1 is inverted. Parameters which are not explicitly shown do not change.

RC1 $X(eq(bit_2(f_2), p_1)/ready_1, \mathrm{f}/rec_1, dat(f_2)/e_1, inv(q_1)/q_1, read/st_2)$
$rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \wedge eq_{del}(st_2)$

RC2 $X(eq(bit_2(f_2), p_1)/ready_1, read/st_2)$
$\neg(rec_1 \wedge eq(bit_1(f_2), inv(q_1))) \wedge eq_{del}(st_2)$

RC3 $X(\mathrm{f}/sts_1)$
$(\neg eq(p_1, q_2) \wedge rec_2) \vee (eq(p_2, q_1) \wedge \neg ready_2) \wedge eq_{read}(st_1) \wedge sts_1$

RC4 $X(del/st_1)$
$eq_{choice}(st_1)$

RC5 $X(\mathrm{t}/sts_1, \langle d_1, p_1, q_1\rangle/f_1, choice/st_1)$
$eq_{read}(st_1) \wedge \neg sts_1$

RC6 $X(read/st_1, eq(bit_2(f_1), p_2)/ready_2, \mathrm{f}/rec_2, dat(f_1)/e_2, inv(q_2)/q_2)$
$rec_2 \wedge eq(bit_1(f_1), inv(q_2)) \wedge eq_{del}(st_1)$

RC7 $X(read/st_1, eq(bit_2(f_1), p_2)/ready_2))$
$\neg(rec_2 \wedge eq(bit_1(f_1), inv(q_2))) \wedge eq_{del}(st_1)$

RC8 $X(\mathrm{f}/sts_2)$
$(\neg eq(p_2, q_1) \wedge rec_1) \vee (eq(p_1, q_2) \wedge \neg ready_1) \wedge eq_{read}(st_2) \wedge sts_2$

RC9 $X(del/st_2)$
$eq_{choice}(st_2)$

RC10 $X(\mathrm{t}/sts_2, \langle d_2, p_2, q_2\rangle/f_2, choice/st_2)$
$eq_{read}(st_2) \wedge \neg sts_2$

Here dat, bit_1, bit_2 are functions yielding values d, p and q, respectively, when applied to a frame $\langle d, p, q\rangle$. The equality functions eq yield value t when applied to identical arguments, and f otherwise. Functions like eq_{del} yield value t when applied to an argument which is identical to *del*, and f otherwise. The booleans come with the usual set of boolean operators $(\neg, \vee, \wedge)$. The bits have only one operator, inversion (inv), which explains the distinction between bits and booleans.

The conditional recursive calls RC1–10 define a directed graph in the following way. Let $X(\vec{x})$ be a state of the one-bit sliding window protocol. For any of the recursive calls RC1–10, if the condition of the recursive call holds in $\vec{x}$, then there exists an edge from $X(\vec{x})$ to $X(\vec{y})$, where $X(\vec{y})$ is the state of the recursive call. There are no other edges than those defined in this way.

4 Informal proof of well-foundedness

The informal proof of the well-foundedness of the graph is greatly simplified by identifying two loosely coupled sets of recursive calls. Observe that the parameters $ready_1$, rec_1, q_1, st_2, sts_2, f_2 are affected only by the recursive calls RC1,2,8–10. Symmetrically, parameters $ready_2$, rec_2, q_2, st_1, sts_1, f_1 are affected only by RC3–7. Observe furthermore that all other parameters either are not affected by any recursive call (the p_i's), or do not occur in any condition (the d_i's, e_i's and $dat(f_i)$'s). Unfortunately, these two sets of recursive calls are not completely independent of each other: q_2 occurs in conditions of RC1,2,8–10, and, symmetrically, q_1 in conditions of RC3–7. There is no other overlap.

For a moment we restrict ourselves to the well-foundedness of a subgraph, namely the one defined by RC3–7. It is helpful to partition the state space in six classes determined by the pair (st_1, sts_1). In all classes but two (those with $eq_{del}(st_1)$) there is at most one recursive call possible. If $eq_{read}(st_1) \wedge sts_1$ then at most RC3 is possible, bringing us in the class $eq_{read}(st_1) \wedge \neg sts_1$. Now only RC5 is possible, bringing us in the class $eq_{choice}(st_1) \wedge sts_1$. If $eq_{choice}(st_1)$, then RC4 brings us to $eq_{del}(st_1)$, in which either RC6 or RC7 is possible, depending on the boolean $rec_2 \wedge eq(bit_1(f_1), inv(q_2))$. Here we have the possibility of a loop, since as well RC6 as RC7 brings us back to $eq_{read}(st_1)$. The situation is depicted in Figure 1. It should be remarked that this figure represents an abstraction of the graph on the state space, and not the graph itself. Transitions in the state space correspond by projection with transitions in the figure, but the cycles in the figure do not correspond to cycles or infinite paths in the original graph. On the contrary: by taking the conditions into account the cycles in the figure are used to exclude infinite paths in the original graph.

Inspection of RC6 shows that the left loop in Figure 1 can occur only once: only in case rec_2, and after RC6, by f/rec_2, we have $\neg rec_2$ for once and for all (no other recursive call affects rec_2). By symmetry RC1 can also occur only once. Furthermore, RC5 is on both the left and the right loop in Figure 1.

The right loop via RC7 is a bit more complicated. We first make the following observation. After recursive call RC5, by $\langle d_1, p_1, q_1 \rangle / f_1$, we have $eq(bit_1(f_1), p_1)$ for once and for all. In this situation, the condition of RC7 implies $\neg(rec_2 \wedge eq(p_1, inv(q_2)))$. This means that in the condition of RC3 the disjunct $(eq(p_2, q_1) \wedge \neg ready_2)$ must be true to enable this recursive call. However, by $eq(bit_2(f_1), p_2)/ready_2$ in RC7, looping would be prevented if $eq(bit_2(f_1), q_1)$.

We are now in a position to complete our argument. Assume the original graph contains an infinite path, say P. Then either infinitely many recursive calls RC3–7, or infinitely many recursive calls RC1,2,8–10 are executed along P. Let us assume infinitely many calls RC3–7, as the other case is fully symmetric. Recall that RC1 is executed at most once. As RC5 has infinitely many occurrences along P, we may choose an occurrence of RC5 after which no RC1 occurs in P. Since RC1 is the only call which affects q_1, q_1 does not change after

the abovementioned occurrence of RC5, and we have both $eq(bit_1(f_1), p_1)$ and $eq(bit_2(f_1), q_1)$ for once and for all. As also RC6 is executed at most once along P, we must encounter an occurrence of RC7 after the abovementioned occurrence of RC5. Now we arrive at a contradiction using the argument from the previous paragraph. It follows that the original graph cannot contain an infinite path.

There exists a function g from the state space to natural numbers which decreases along the paths. This follows from the analysis above. Conversely, given such a function, it follows that the graph does not contain an infinite path. We give such a function below. The reader can easily verify that it has the desired property. To find such a function is not so easy. The function below is bounded by 28, which means that the graph does not contain any path longer than that.

Let $(x_1, x_2, x_3, x_3', x_4, x_4')^i$ abbreviate (with $if(b, x, y) = x$ if $b = \mathrm{t}$, and y if $b = \mathrm{f}$)

$$if(eq_{read}(st_i), if(sts_i, x_1, x_2), if(eq_{choice}(st_i), if(sts_i, x_3, x_3'), if(sts_i, x_4, x_4'))).$$

Define $g(ready_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, ready_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2)$ by

$$\begin{aligned}
&if(rec_1, 8, 0)+\\
&if(rec_2, 8, 0)+\\
&if(rec_2 \wedge \neg eq(p_1, q_2),\\
&\quad if(eq(bit_1(f_1), p_1), (4,3,2,5,1,4)^1, (4,3,6,5,5,4)^1),\\
&\quad if(eq(bit_2(f_1), q_1), (if(\neg ready_2 \wedge eq(p_2, q_1), 4, 0), 3, 2, 5, 1, 4)^1, (4,3,6,5,5,4)^1))+\\
&if(rec_1 \wedge \neg eq(p_2, q_1),\\
&\quad if(eq(bit_1(f_2), p_2), (4,3,2,5,1,4)^2, (4,3,6,5,5,4)^2),\\
&\quad if(eq(bit_2(f_2), q_2), (if(\neg ready_1 \wedge eq(p_1, q_2), 4, 0), 3, 2, 5, 1, 4)^2, (4,3,6,5,5,4)^2)).
\end{aligned}$$

5 Resolution proof of well-foundedness

In this section we describe the experiment of proving the graph well founded using OTTER [5, 6, 8]. In this case the predicate WF has the following first order definition:

$$\forall \vec{x}\ ((\bigwedge_{i=1}^{10}(C_i(\vec{x}) \rightarrow WF(e_i(\vec{x})))) \rightarrow WF(\vec{x})) \tag{3}$$

Here $\vec{x}$ is the parameter list of X, i.e. a 18-tuple, $C_i(\vec{x})$ is the condition of the i-th recursive call, and $e_i(\vec{x})$ is the parameter of the i-th recursive call. For example, if $i = 1$ then $C_i(\vec{x})$ is the condition of RC1,

$$rec_1 \wedge eq(bit_1(f_2), inv(q_1)) \wedge eq_{del}(st_2),$$

and $e_i(\vec{x})$ is the parameter of RC1,

$$\begin{aligned}
&e_i(ready_1, rec_1, sts_1, d_1, e_1, p_1, q_1, f_1, st_1, ready_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, st_2) =\\
&(eq(bit_2(f_2), p_1), \mathrm{f}, sts_1, d_1, dat(f_2), p_1, inv(q_1), f_1, st_1, ready_2, rec_2, sts_2, d_2, e_2, p_2, q_2, f_2, read).
\end{aligned}$$

We have to prove $\forall \vec{x}\, WF(\vec{x})$. The standard way to do this by resolution is to clausify the above definition extended with $\neg \forall \vec{x}\, WF(\vec{x})$ and to look for a derivation of the empty clause. The idea behind the refutation of $\neg\, WF(\vec{c})$ (with $\vec{c}$ Skolem constants) is that from an arbitrary

vertex $\vec{c}$ paths $\vec{c}, e_i(\vec{c}), e_j(e_i(\vec{c})), \ldots$ are examined along which $C_i(\vec{c}), C_j(e_i(\vec{c})), \ldots$ must hold. Such a path ends when the corresponding conjunction of conditions is found contradictory. This finally yields the empty clause if all paths are finite. The potential advantage over the combinatorial techniques lies in the fact that the inferred contradiction may be proved for a large part of the vertex set without having to consider individual vertices. For example, $\neg(even(n) \wedge even(n+1))$ in the introduction can be proved for all $n \in \mathbb{N}$.

5.1 Clausification

A direct translation into clauses of the definition of *WF* above leads to an unmanageable set of clauses. The reason is the conjunction of length 10 in the antecedent of the definition. The informal proof from the previous section suggests a partition of the state space in 36 parts, each determined by the value of the quadruple $(sts_1, st_1, sts_2, st_2)$. Given a value of $(sts_1, st_1, sts_2, st_2)$, at most 4 of the conditions $C_i(\vec{x})$ can be true. The cases in which 4 conditions $C_i(\vec{x})$ can be true are all of the form (sts_1, del, sts_2, del). In all other cases at most 3 conditions $C_i(\vec{x})$ can be true. Applying this idea leads to a manageable set of about 370 clauses. As an example, we give the part of the definition of *WF* in the case $(\mathrm{t}, read, sts_2, del)$.

```
(all ready1 (all rec1           (all d1 (all e1 (all p1 (all q1 (all f1
(all ready2 (all rec2 (all sts2 (all d2 (all e2 (all p2 (all q2 (all f2
(%RC3
((((-biteq(p1,q2) & -booleq(rec2,false)) |
   (-biteq(p2,inv(q1)) & -booleq(ready2,true))) ->
             WF(ready1,rec1,false,d1,e1,p1,q1,f1,read,
                ready2,rec2, sts2,d2,e2,p2,q2,f2,del))
&%RC1,2
(((-booleq(rec1,false) & -biteq(bit1(f2),q1)) ->
((-biteq(p1,inv(bit2(f2))) ->
               WF(true,false,true,d1,dat(f2),p1,inv(q1),f1,read,
                 ready2,rec2,sts2,d2,e2,     p2,q2,     f2,read))
        &
 (-biteq(p1,bit2(f2)) ->
               WF(false,false,true,d1,dat(f2),p1,inv(q1),f1,read,
                  ready2,rec2,sts2,d2,e2,     p2,q2,     f2,read))))
                &
 ((-booleq(rec1,true) | -biteq(bit1(f2),inv(q1))) ->
((-biteq(p1,inv(bit2(f2))) -> WF(true,rec1,true,d1,e1,p1,q1,f1,read,
                                 ready2,rec2,sts2,d2,e2,p2,q2,f2,read))
        &
 (-biteq(p1,bit2(f2)) -> WF(false,rec1,true,d1,e1,p1,q1,f1,read,
                            ready2,rec2,sts2,d2,e2,p2,q2,f2,read))))))

-> %RC3|RC1,2

WF(ready1,rec1,true,d1,e1,p1,q1,f1,read,
   ready2,rec2,sts2,d2,e2,p2,q2,f2,del))))))))))))))))).
```

There are several things which have to be explained here. First, we have introduced equality predicates for booleans and bits, axiomatized by the following clauses.

```
(all b booleq(b,b)).
-booleq(false,true).
-booleq(true,false).
(all b (booleq(b,true) | booleq(b,false))).
(all b (-booleq(b,true) | -booleq(b,false))).

(all b biteq(b,b)).
(all b all b1 (biteq(b,b1) | biteq(b,inv(b1)))).
(all b all b1 (-biteq(b,b1) | -biteq(b,inv(b1)))).
```

This axiomatization is far from complete. For example, the axioms for symmetry, transitivity and substitutivity are missing. However, the definition of *WF* has been formulated in such a way that the above clauses are sufficiently complete for our application.

Second, we have reformulated the conditions in a negative way. For example, rec_1 from the condition of RC1 has become `-booleq(rec1,false)`. To understand this, recall the definition of *WF*:

$$\forall \vec{x}\ ((\bigwedge_{i=1}^{10}(C_i(\vec{x}) \rightarrow WF(e_i(\vec{x})))) \rightarrow WF(\vec{x})).$$

The conditions $C_i(\vec{x})$ occur on positive positions, and this polarity is preserved during the clausification. Reformulating the conditions in a negative way has the effect that the resulting clauses contain only one positive literal, namely the *WF* literal stemming from the consequent $WF(\vec{x})$ in the above definition. This facilitates the use of (negative) hyperresolution.

Third, the following rewrite rules are necessary in the presence of the rudimentary equality axioms above.

```
list(demodulators).
(inv(inv(x)) = x).
(bit1(frame(y,x1,x2)) = x1).
(bit2(frame(y,x1,x2)) = x2).
(dat(frame(y,x1,x2)) = y).
end_of_list.
```

Here `frame` is a frame constructor, which does not show up in the clause `RC3|RC1,2`, but does in clauses stemming from RC5 and/or RC10. There $\langle d, p, q\rangle$ is translated into `frame(d,p,q)`.

Fourth, the assignment $eq(bit_2(f_2), p_1)/ready_1$ is modelled as a case distinction:

```
(-biteq(p1,inv(bit2(f2))) -> WF(true,...))
&
(-biteq(p1,bit2(f2)) -> WF(false,...))
```

Note that in literals like `-biteq(p1,bit2(f2))` the p-like arguments such as `p1` (`p2`, `bit1(..)`) always precede q-like arguments like `bit2(f2)` (`q1`, `q2`). This has been done systematically, so that we do not need the symmetry axiom here.

5.2 Refutation

As a consequence of the partitioning of the state space applied in the previous subsection we cannot simply refute $\neg\forall\vec{x}\,WF(\vec{x})$, since the Skolemization of this formula would introduce Skolem constants different from `true, false, read, del, choice`. We could replace $WF(\vec{x})$ by a suitable conjunction of 36 conjuncts, one for each value of the quadruple $(sts_1, st_1, sts_2, st_2)$, and refute the universal closure of the conjunction. However, it is more attractive to refute the conjuncts one by one. Thus a problem which may be too difficult is decomposed into 36 not too difficult problems. (Divide and conquer!) Moreover, the solutions of solved subproblems can be used to solve new subproblems.

The next problem to tackle is: in which order do we solve the subproblems. It turned out to be false that the 36 subproblems were all 'not too difficult'. The informal analysis from the previous section suggests that the case $(\mathsf{f}, read, \mathsf{f}, read)$ is the simplest one. The reason for this is that now only RC5 and RC10 are possible, and both are instantiating the frame parameters. This means that the values of $bit_i(f_j)$ become expressed in terms of other parameters, which reduces the total number of parameters. More evidence is provided by the crucial role that RC5 (respectively RC10) plays in the informal argument in the previous section. Finally, in the definition of g from the previous section, the variable x_2 in $(x_1, x_2, x_3, x_3', x_4, x_4')^i$ represents the case $\neg sts_i \wedge eq_{read}(st_i)$ and has the lowest maximum in the definition of g. Similar considerations lead to the following order of subproblems: first all 4 subproblems with only status *read*, then the 8 subproblems with one status *read* and one *del*, then the 8 subproblems with one status *read* and one *choice*, then the remaining 16 subproblems with no status *read*.

In order to solve the first subproblem we added the following formula to the set of input formulas from the previous subsection.

```
formula_list(sos).
-(all ready1 (all rec1 (all d1 (all e1 (all p1 (all q1 (all f1
 (all ready2 (all rec2 (all d2 (all e2 (all p2 (all q2 (all f2
WF(ready1,rec1,false,d1,e1,p1,q1,f1,read,
   ready2,rec2,false,d2,e2,p2,q2,f2,read)
)))))))))))).
end_of_list.
```

The resulting set of clauses is refuted using negative hyperresolution in about half an hour on a SPARC station 10 with sufficient main memory. This solves the first subproblem. Thereafter the negation sign is deleted from the formula above and the resulting formula is added to the set of input formulas from the previous subsection. Then the next subproblem is solved and added, and so on. Later refutations are considerably faster, since they can use the previous solutions. For example, the subproblem corresponding to the case $(\mathsf{t}, read, \mathsf{t}, read)$ is solved instantaneously when using the solution to the other 3 subproblems with only status *read*. (This can easily be understood by inspection of RC3 and RC8.) The whole run of all 36 subproblems takes less than 3 hours.

5.3 Optimization

We have been experimenting with other strategies. As can be expected, binary resolution is far less efficient than hyperresolution. Unit resulting resolution gives a slight improvement when added to negative hyperresolution, but only if the generation of positive *WF* literals is suppressed (e.g. by `assign(max_distinct_vars,0)`).

Another possibility for optimization is weighting. The value of the `max_weight` parameter of OTTER is important, a sharp value can save a factor 5 in some cases. The timings above are with a sharp value of `max_weight`. With clever weighttemplates it is possible to save a factor 3 in some cases. The idea is that the Skolem constants introduced during the clausification of the negative formula in the set of support above are the 'variables' of the problem. During the examination of a path they are replaced by `true, false, inv(q1)` and so on. This is a kind of instantiation and we can give 'instantiated' *WF* literal a lighter weight then those with 'variables', so that OTTER gives some priority to clauses containing *WF* literals representing vertices which are at some distance of the starting point.

A third possibility for optimization is rewriting. By simply substituting `eqbool` for `booleq` and `eqbit` for `biteq` everywhere in the input, OTTER treats equality predicates as real equalities (satisfying the equality axioms) and not as arbitrary predicates. With the appropriate options set, this includes dynamic rewriting. The improvement in performance is modest, probably since the first positive equality literal that can be used for rewriting is inferred only halfway the proof seach. We expect more improvement when more of the evaluation of the conditions is handled by rewriting.

6 Conclusion

We proposed and investigated a symbolic approach to proving the well-foundedness of a graph. This approach complements combinatorial algorithms for finite graphs (notably acyclicity tests), since it also applies to infinite graphs. Even in the finite case the symbolic approach may be fruitfully applied in cases in which the graph has a symbolic definition. We tested the symbolic approach on a non-trivial graph. The graph has a symbolic definition, is essentially finite (about 155.000 vertices, to be multiplied by $|D|^6$, where $|D|$ is the size of data set D) and is proved well founded by the resolution theorem prover OTTER in less than 3 hours on a SPARC station 10 with sufficient main memory.

Acknowledgement

We thank Bill McCune for on-line assistance with OTTER.

References

[1] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. Technical Report 98, Logic Group Preprint Series, Utrecht University, 1993 (Proceedings CONCUR'94, to appear).