# A Proof Assistant for Symbolic Model-Checking

J. C. Bradfield
Laboratory for Foundations of Computer Science
University of Edinburgh
The King's Buildings
EDINBURGH
U.K. EH9 3JZ
email: jcb@dcs.ed.ac.uk

**Abstract:** We describe a prototype of a tool to assist in the model-checking of infinite systems by a tableau-based method. The tool automatically applies those tableau rules that require no user intervention, and checks the correctness of user-applied rules. It also provides help with checking the well-foundedness conditions required to prove liveness properties. The tool has a general tableau-manager module, and may use different reasoning modules for different models of systems; a module for Petri nets has been implemented.

## 1 Introduction.

Many approaches to the verification of systems have been developed in recent years; model-checking is one to which both theoretical effort and implementation effort have been devoted. The principle of model-checking as it was first studied ([ClE81], [CES86]) is that one expresses a desirable property in some suitable temporal logic, and then checks this property algorithmically. However, there are drawbacks to model-checking. The first is that typically model-checking algorithms traverse the global state-space of the system under investigation, whereas often this may not be necessary. To address this problem, Stirling and Walker [StW89] introduced a tableau-based technique for local model-checking, in which parts of the system are not explored until information on their local structure is required. The second problem follows from the first, and is that traditional model-checking cannot deal with infinite-state systems, although many interesting systems are infinite or potentially infinite. Clearly, model-checking arbitrary infinite systems can no longer be algorithmic, so Stirling and the present author extended tableau-based model-checking to provide a general technique for infinite systems; knowledge about the system may be used where necessary to supplement the model-independent rules. (See [BrS90], [Bra91a], [Bra91b], [BrS92].)

Our infinite-state model-checking technique is well suited to computer-aided verification: although in general one may need powerful reasoning, there is a large part of the technique that is more or less automatic. Moreover, some form of computer assistance is essential in practice, since only very simple formulae and systems can be done by hand. This paper describes a prototype of a proof assistant for our system; the prototype has a module for reasoning with Petri nets, but other models may also be used. The implementation is closely tied to the theoretical basis, and even in prototype has given useful feedback on topics such as how finitely to represent infinite systems and how to represent the information needed to prove well-foundedness (which we require for the proof of liveness properties (in general, least fixed-points in the modal mu-calculus)). The tool provides the user with a reasonably simple X Window System interface for constructing tableaux, and, more importantly, checks the correctness of all rules used in the tableau—this is not a trivial operation, since we are dealing with infinite sets of system states. It can also build a tableau automatically up to the point where user intervention is required to resolve a choice. Moreover, the tool also provides some help with proving the required well-foundedness relations. These relations are defined by the tableau, and are typically complex; however, there is often a simpler relation, apparent to the intelligent observer or designer, the well-foundedness of which implies the well-foundedness of the tableau relations. In such cases the tool can verify that this implication is valid, so that the user only has to check the well-foundedness of the simple relation.

## 2 The modal mu-calculus.

The temporal logic we use is the propositional modal mu-calculus, introduced by Kozen [Koz83]. This is a very rich logic which is now increasingly popular, both as an object of study ([KoP83], [EmL86], [StE89]) and as a logic for use in verification problems and tools such as the Concurrency Workbench ([CPS89], [BrA91], [Bra91b]).

The formulae have the form

$$\Phi \ ::= \ Z \mid \neg\Phi_1 \mid \Phi_1 \wedge \Phi_2 \mid [K]\Phi_1 \mid \nu Z.\Phi_1$$

where $Z$ ranges over a countable set Var of variables, $K$ ranges over subsets of a label set $\mathscr{L}$, and $\nu Z.\Phi_1$ is subject to the restriction that any free occurrence of $Z$ in $\Phi_1$ must be within the scope of an even number of negation symbols. The usual boolean and modal dual connectives are employed ($\langle K\rangle\Phi \stackrel{\text{def}}{=} \neg[K]\neg\Phi$), and a dual to $\nu$ is defined by

$$\mu Z.\Phi \stackrel{\text{def}}{=} \neg\nu Z.\neg\Phi[\neg Z/Z]$$

where $\Phi[\Psi/Z]$ denotes the syntactic substitution of $\Psi$ for free occurrences of $Z$ in $\Phi$. The symbol $\sigma$ is used to range over $\nu$ and $\mu$. The following abbreviations are used: $[-K]\Phi \stackrel{\text{def}}{=} [\mathscr{L} - K]\Phi$; $[a_1, \ldots, a_n]\Phi \stackrel{\text{def}}{=} [\{a_1, \ldots, a_n\}]\Phi$; $[-]\Phi \stackrel{\text{def}}{=} [\mathscr{L}]\Phi$; and similarly for $\langle K\rangle$.

A model for this logic is a labelled transition system $\mathscr{T} = (\mathscr{S}, \{ \xrightarrow{a} \mid a \in \mathscr{L} \})$ (where $\mathscr{S}$ is a set of states) together with a valuation $\mathscr{V}: \mathrm{Var} \to 2^{\mathscr{S}}$. The denotation $\|\Phi\|^{\mathscr{T}}_{\mathscr{V}}$ (or $\|\Phi\|_{\mathscr{V}}$ when $\mathscr{T}$ is understood) of a formula is a set of states given by the usual rules for variables and booleans together with the modal and fix-point rules

$$\|\nu Z.\Phi\|_{\mathscr{V}} = \bigcup \{ S \subseteq \mathscr{S} \mid \|\Phi\|_{\mathscr{V}[Z:=S]} \supseteq S \}$$

$$\|[K]\Phi\|_{\mathscr{V}} = \{ s \in \mathscr{S} \mid \forall s' \in \mathscr{S} \,.\, \forall a \in K \,.\, s \xrightarrow{a} s' \;\Rightarrow\; s' \in \|\Phi\|_{\mathscr{V}} \}$$

where $\mathscr{V}[Z := S]$ is the valuation $\mathscr{V}'$ which agrees with $\mathscr{V}$ save that $\mathscr{V}'(Z) = S$.

## 3 The tableau system.

We now describe the tableau system for model-checking. This description is necessarily brief; see [BrS90] or [Bra91a] for more information, and [Bra91b] or [BrS92] for a full account.

In order to deal with the fix-point formulae, we extend the mu-calculus by constants $U, V, \ldots$, and introduce definition lists of the form $\Delta = (U_1 = \Phi_1, \ldots, U_n = \Phi_n)$ which assign formulae to constants (subject to the constraint that the only constants appearing in $\Phi_i$ are those in $\{U_1, \ldots, U_{i-1}\}$), and write $\Delta(U)$ for the formula assigned to $U$ in $\Delta$. The denotation $\|\Phi_\Delta\|_{\mathscr{V}}$ of a formula with respect to $\Delta$ is given in the natural way by

$$\|\Phi_{\Delta\cdot(U=\Psi)}\|_{\mathscr{V}} = \|\Phi_\Delta\|_{\mathscr{V}[U:=\|\Psi_\Delta\|_{\mathscr{V}}]}.$$

The sequents of our tableaux have the form $S \vdash_\Delta \Phi$ where $S$ is a set of states, $\Phi$ is a formula of the extended mu-calculus in positive normal form (that is, using the dual operators and De Morgan laws to move negations inwards so that they only apply to variables), and $\Delta$ is a definition list. These sequents are treated as goals to be proved—$S \vdash_\Delta \Phi$ means we want to prove that $S \subseteq \|\Phi_\Delta\|_{\mathscr{V}}$—and tableaux are built up by means of the following goal-directed rules (where $s \xrightarrow{K} s'$ means $\exists a \in K \,.\, s \xrightarrow{a} s'$).

$\wedge$ $\qquad \dfrac{S \vdash_\Delta \Phi_1 \wedge \Phi_2}{S \vdash_\Delta \Phi_1 \qquad S \vdash_\Delta \Phi_2}$

$\vee$ $\qquad \dfrac{S \vdash_\Delta \Phi_1 \vee \Phi_2}{S_1 \vdash_\Delta \Phi_1 \qquad S_2 \vdash_\Delta \Phi_2}$ $\qquad S = S_1 \cup S_2$

$[K]$ $\qquad \dfrac{S \vdash_\Delta [K]\Phi}{S' \vdash_\Delta \Phi}$ $\qquad S' = \{ s' \mid \exists s \in S \,.\, s \xrightarrow{K} s' \}$

$\langle K \rangle$ $\qquad \dfrac{S \vdash_\Delta \langle K \rangle\Phi}{f(S) \vdash_\Delta \Phi}$ $\qquad f: S \to f(S)$ s.t. $\forall s \in S \,.\, s \xrightarrow{K} f(s)$

| | | |
|---|---|---|
| $\sigma Z.$ | $\dfrac{S \vdash_\Delta \sigma Z.\Phi}{S \vdash_{\Delta'} U}$ | $U$ not in $\Delta$ and $\Delta' = \Delta \cdot (U = \sigma Z.\Phi)$ |
| Un | $\dfrac{S \vdash_\Delta U}{S \vdash_\Delta \Phi[U/Z]}$ | $\Delta(U) = \sigma Z.\Phi$ |
| Thin | $\dfrac{S \vdash_\Delta \Phi}{S' \vdash_\Delta \Phi}$ | $S' \supseteq S$ |

The function $f$ in the $\langle K \rangle$ rule is a choice function whose purpose is to choose the successor state that most quickly leads towards 'termination', in a sense made precise shortly.

One keeps applying rules to the leaves of the tableau (starting with a root sequent for the property one wishes to prove), until all leaves are *terminal.* A leaf is terminal if no rule other than Thin applies, that is

(i) $\Phi = Z$ or $\Phi = \neg Z$, or

(ii) $\Phi = \langle K \rangle \Psi$ and $\exists s \in S \,.\, \forall s' \in \mathscr{S} \,.\, \text{not } s \xrightarrow{K} s'$,

or the state set is trivial

(iii) $S = \varnothing$

or finally

(iv) $\Phi = U$ and $\Delta(U) = \sigma Z.\Psi$ and $\mathbf{n}$ has an ancestor node $\mathbf{n}' = S' \vdash_{\Delta'} U$ such that $S' \supseteq S$ (in which case we call $\mathbf{n}$ a $\sigma$-terminal and $\mathbf{n}'$ its companion).

A terminal node is deemed to be *successful* iff

- it is in class (i) and is true, i.e. if $\Phi = Z$ then $S \subseteq \mathscr{V}(Z)$ and if $\Phi = \neg Z$ then $S \cap \mathscr{V}(Z) = \varnothing$; or
- it is in class (iii); or
- it is in class (iv) with $\sigma = \nu$; or
- it is in class (iv) with $\sigma = \mu$ and satisfies the **mu-success** conditions defined below.

A tableau is successful if it is finite and all its terminals are successful.

For the correctness of least fix-points, we define a relation on the states at companion nodes, which corresponds to the dependency of truth of a least fix-point at one state on the truth at other states; then we need to ensure that there are no infinite dependency chains for least fix-points. Since we must take account of nested fix-points, this relation is quite complicated. We start by defining paths and extended paths in the tableau:

In a tableau, a *path* from a state $s$ at a node $\mathbf{n}$ to a state $s'$ at a node $\mathbf{n}'$ is a sequence $(s, \mathbf{n}) = (s_0, \mathbf{n}_0), (s_1, \mathbf{n}_1), \ldots, (s_k, \mathbf{n}_k) = (s', \mathbf{n}')$ of states and nodes such that

- $\mathbf{n}_{i+1}$ is a child of $\mathbf{n}_i$;
- if $\mathbf{n}_i = S_i \vdash_{\Delta_i} \Phi_i$ then $s_i \in S_i$;
- if the rule applied to $\mathbf{n}_i$ is $[K]$ then $s_i \xrightarrow{K} s_{i+1}$, if the rule is $\langle K \rangle$ then $s_{i+1} = f(s_i)$, and otherwise $s_{i+1} = s_i$.

We write $s@\mathbf{n} \mathrel{\underset{\cdot}{\longrightarrow}} s'@\mathbf{n}'$ if there is a path from $s$ at $\mathbf{n}$ to $s'$ at $\mathbf{n}'$.

Now there is an *extended path* from $s$ at $\mathbf{n}$ to $s'$ at $\mathbf{n}'$, written $s@\mathbf{n} \mathrel{\overset{\cdot}{\underset{\cdot}{\longrightarrow}}} s'@\mathbf{n}'$, if either

(i) $s@\mathbf{n} \mathrel{\underset{\cdot}{\longrightarrow}} s'@\mathbf{n}'$, or

(ii) there is a node $\mathbf{n}'' = S'' \vdash_{\Delta''} U$ and a finite sequence of states $s_0, s_1, \ldots, s_k$ and nodes $\mathbf{n}_1, \ldots, \mathbf{n}_k$ for $k \geq 0$, where each $\mathbf{n}_i$ is a terminal with companion $\mathbf{n}''$, such that $s@\mathbf{n} \mathrel{\underset{\cdot}{\longrightarrow}} s_0@\mathbf{n}''$ and $s_i@\mathbf{n}'' \mathrel{\overset{\cdot}{\underset{\cdot}{\longrightarrow}}} s_{i+1}@\mathbf{n}_{i+1}$ for $0 \leq i < k$, and $s_k@\mathbf{n}'' \mathrel{\overset{\cdot}{\underset{\cdot}{\longrightarrow}}} s'@\mathbf{n}'$.

Finally, we can define the relation on states, and the well-foundedness condition:

Let $\mathbf{n} = S \vdash_{\Delta} U$ be a $\mu$-terminal with companion node $\mathbf{n}' = S' \vdash_{\Delta'} U$. Define a relation $\sqsupset_{\mathbf{n}'}$ on $S'$ by $s \sqsupset_{\mathbf{n}'} s'$ iff $s@\mathbf{n}' \mathrel{\overset{\cdot}{\underset{\cdot}{\longrightarrow}}} s'@\mathbf{n}''$ for any terminal $\mathbf{n}''$ whose companion is $\mathbf{n}'$. The condition **mu-success** is that $\sqsubset_{\mathbf{n}'}$ should be well-founded, that is, there should be no infinite chains $s_0 \sqsupset_{\mathbf{n}'} s_1 \sqsupset_{\mathbf{n}'} \cdots$.

If now we have a successful tableau, then the root sequent is true, and if the root sequent is true, then a successful tableau exists (see [Bra91b] or [BrS92] for the proofs).

For the purposes of our prototype, two small modifications are made to the tableau rules. The first is that we do not have a Thin rule, but instead allow implicit thinning in every rule, so that, for example, the $\vee$ rule side-condition is $S \subseteq S_1 \cup S_2$. Implicit thinning saves space and time in the tableau; explicit thinning will be added when convenient. The second change is specific to the Petri net model class: in the choice function at a $\langle K \rangle$ rule; rather than specifying the successor marking of $M$, we specify the transition to fire. This is both more natural and easier to represent.

## 4 Petri nets.

For our purposes, a Petri net is a place/transition net, so that formally a net $\mathscr{N} = (\mathbf{S}, \mathbf{T}; \mathbf{F})$ comprises two disjoint sets $\mathbf{S}$ (the *places*) and $\mathbf{T}$ (the *transitions*) together with a map $\mathbf{F}\colon (\mathbf{S} \times \mathbf{T}) \uplus (\mathbf{T} \times \mathbf{S}) \to \mathbb{N}$ (where $\uplus$ is disjoint union). $\mathbf{F}$ gives the weight $w$ of an arc connecting a place and a transition: if $\mathbf{F}(s,t) = w$ then when $t$ fires it removes $w$ tokens from place $s$, and if $\mathbf{F}(t,s) = w$ it adds $w$ tokens to $s$. A *marking* of $\mathscr{N}$ is a map $M\colon \mathbf{S} \to \mathbb{N}$, giving the number of tokens in each place. As usual, ${}^\bullet x$ is the map $\mathbf{S} \uplus \mathbf{T} \to \mathbb{N}$ given by ${}^\bullet x(y) = \mathbf{F}(y,x)$, and similarly $x^\bullet(y) = \mathbf{F}(x,y)$. The behaviour of the net viewed as a transition system with markings as states and transitions (of the net) as labels is given by the normal token game rules: $M \xrightarrow{t} M'$ iff ${}^\bullet t \leq M$ and $M' = M - {}^\bullet t + t^\bullet$.

To apply the tableau system to nets, we need to make several decisions. Firstly, we must decide what our 'atomic propositions' are, that is, what predefined variables we want in order to express properties of nets. An obvious choice is to take linear (in)equalities on places, so that for places $s_i$ and integers $a_i$ and $b$, the expression $a_1 s_1 + \cdots + a_n s_n \leq b$ is deemed to be a variable whose denotation is those markings $M$ satisfying the inequality $a_1 M(s_1) + \cdots + a_n M(s_n) \leq b$.

Secondly, we need a means of representing the (infinite) sets of markings on the left of sequents. As considered by [Bra91b], here again (boolean combinations of) linear (in)equalities are often sufficient, and since they are also not too difficult to reason with, they were adopted in our tool.

For example, we could consider the net

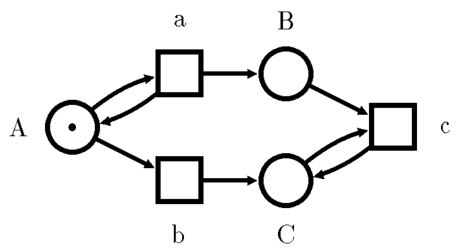


Figure 1

and the formulae $\nu Z.(\mathrm{A} = 1) \wedge \langle - \rangle Z$ (some infinite execution exists on which A is always 1) and $\mu Y.\nu Z.[\mathrm{c}]Y \wedge [-\mathrm{c}]Z$ (event c happens only finitely often). The first of these formulae will demonstrate the basic use of the tool without least fix-points; the second will demonstrate the tool's facilities for checking well-foundedness.

## 5 Tableau construction with the tool.

To construct a tableau using our tool, one starts up the tool, specifying the model being investigated. The model is specified as an ML data structure; in the case of Petri nets, a simple encoding of the definition. A window is created in which the tableau will be displayed, and above which there are some command buttons. One then clicks the 'Make Root' button, and types the initial state set and formula. The choice of state set is the first implicit thinning; typically one is interested in one initial state, but one must thin to a larger set to build a tableau. The logical notation is adapted slightly for the ASCII character set and ease of parsing: `&` and `|` are used for $\wedge$ and $\vee$, $\mu$ and $\nu$ are written as `mu` and `nu`, and 'atomic propositions' are enclosed in braces, thus: `{A + C = 1}`.

Figure 2 shows the main window of the tool just after the initial sequent has been typed in, using the net and the first formula given above, with an initial set $\{\mathrm{A} = 1\}$. The reverse video indicates that the node is selected; the border of the sequent is grey, which indicates that this node is not valid. By valid we mean that either it is the premise of a validly applied rule, or that it is a successful terminal, or that it is a $\mu$-terminal. Thus, when all nodes in a tableau are valid, the only remaining task is to check the well-foundedness conditions.


Figure 2

There are eight active command buttons: 'Log' brings up a log of all informational messages issued by the tool in this session; 'Wfedit', 'Wfchk' and 'Auto-apply' will be described later.

The 'Show' button describes the current node, giving information associated with it. For this particular node, there is no information other than the sequent set and formula.

The 'Edit' button can be used to edit the state set on the left of the sequent; this, of course, renders invalid both the edited node and its parent. After editing (or after anything else that invalidates a node), the 'Chk rule' button can be pressed to check the validity of the node. This invokes functions associated with the particular model class and representation of state sets (in this case, Petri nets and linear inequalities) to determine whether the rule applied to the node is a correct instantion of a tableau rule. For example, in the case of an $\wedge$ rule, it is necessary to check that both consequent sets contain the premise set.

The 'Apply' rule applies whatever rule is appropriate—since we do not have an explicit Thin, the rule is entirely determined by the formula. If the rule is anything other than $\vee$ or $\langle K \rangle$, no user input is required: if the node is a leaf, the new children nodes are created and their state sets calculated, and if the node is not a leaf, then the consequent sets are recalculated from the premise. (So if one edits the state set at a node, 'Apply' is used to effect the necessary changes further down the tableau.) In either case, the current node is marked valid.

Figure 3 shows the same tableau after as much as possible has been done without making any choices. Note that the border of the left-hand leaf is black; 'Chk Rule' has been used to check that this terminal is successful, which means simply checking that the state set satisfies the atomic proposition.

To apply the diamond rule to the right-hand leaf, the user must specify a choice function. As mentioned above, we specify for each marking the transition to be fired. This again raises the question of how to represent such functions. For this prototype, we use a simple method: we just give a list $(\mathbf{M}_1, t_1), \ldots, (\mathbf{M}_n, t_n)$

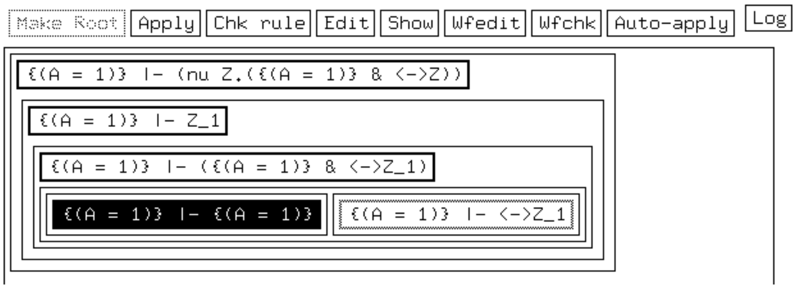


Figure 3

of marking sets (expressed in the same notation used throughout) and transitions, and for a marking $M$, we fire the first $t_i$ such that $M \in \mathbf{M}_i$. The notation used in the tool is $\mathbf{M}_1$ `->` $t_1$ `---` ... `---` $\mathbf{M}_n$ `->` $t_n$. For this particular tableau, all we need do is fire the a transition all the time, so the choice function is `tt -> a`. The tool then calculates the successor set and creates the child node. Figure 4 shows the tableau after applying the diamond rule and 'Show'ing the diamond node.

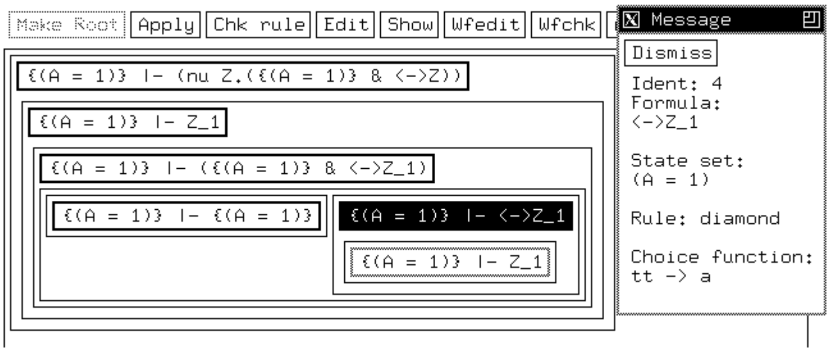


Figure 4

The only remaining task for this tableau is to verify that the new leaf is a successful terminal. 'Chk rule' does this by searching up the tableau to find a companion; in this case it finds the first $Z_1$ node $\mathbf{n}'$, and finds indeed that the state set at the terminal is a subset of the state set at $\mathbf{n}'$.

So far we have assumed that the user is actually getting everything right; what happens when a rule application is incorrect? The answer to this depends on the reasoning power of the module that handles rule checking for the particular class of models. In the present implementation, a (set of) marking(s) that illustrates the bad rule application is given. For example, suppose that we specified a choice function that always chose event c. This would be bad, since c cannot always fire from markings in $\{A = 1\}$. On trying to apply the diamond rule, the message in Figure 5 would be given, and similar messages are given for other incorrect rule applications.

```
Dismiss
Can't apply choice function: chosen transition is blocked for marking:
{A = 1, B = 0}
```

Figure 5

Finally, we should mention $\vee$ rules, and the 'Auto-apply' button. When an $\vee$ rule is applied to a leaf, the user is asked to specify the state set for the left disjunct, and the right state set is calculated from that—it is generally a good idea to have non-intersecting state sets in a disjunction, and sets calculated by the tool have this property. The user can change this by hand-editing the sets, but even then the tool will always take the left disjunct if possible when it calculates the tableau $\sqsupset$ relations.

The 'Auto-apply' button is simply a convenience; it builds the tableau as far as can be done without user intervention. It is particularly useful when one realizes some set high up in the tableau is wrong; after editing it, 'Auto-apply' will propagate the change everywhere below, using existing left disjuncts and choice functions if these still give valid rules.

## 6 Implementation.

We have described the tool as the user sees it; we now say a little about the implementation.

The main body of the tool is written in Standard ML, and deals with the maintenance of tableau data-structures, parsing mu-calculus formulae, etc. (All parsers are written using the ML-Yacc parser generator.) It receives commands from the user, and gives information to the user (e.g. displaying tableaux) via a separate front end.

The X Window System front end is a C program which communicates via a pipe with the main program. This is just a matter of convenience: the main body, of course, neither knows nor cares how its user-interface functions are actually implemented. The front end is the bare minimum necessary for sensible use of the tool and would have to be improved in a production version.

The other significant part of the tool is the module for dealing specifically with Petri nets. Not surprisingly, this is the largest part in lines of code. This module is called by the main module to do all manipulations on sets of markings, so it must be able to handle intersection, union and complementation of marking sets. Since our representation of marking sets is by boolean combinations of linear inequalities, these operations are straightforward. It must also calculate successor sets for the modal rules; these are given by the Petri net firing rules. An important practical point is that during the construction of a tableau, many such operations are performed, and by the time one reaches a leaf, a formula obtained just by applying the rules probably has many copies of each atomic proposition in many positions, most of them redundant. This is highly undesirable, both because it will baffle the user with unnecessary complexity, and because it will increase the time required to manipulate sets. Therefore, considerable effort is devoted to simplifying formulae, using various manipulations to remove redundant subformulae. This is by no means complete—we do not use any exponential algorithms in simplification—but we achieve sufficient simplification to avoid irritating the user excessively!

The other task that the reasoning module has to perform, is to determine whether sets of markings are empty. (More often, it is asked whether one set is contained in another, but since we can complement and intersect sets, this is the same problem.) The technique for doing this is first to reduce the formula representing the set to disjunctive normal form, and then to check that each disjunct is empty. This amounts to showing that a set of linear inequalities has no non-negative integer solution. To do this, we use the symbolic manipulation language Maple; we use a branch and bound integer programming algorithm built around the simplex package supplied with Maple. This has the advantage of giving a minimal counter-example if the set is not empty, and it is this counter-example that is displayed in the error message for invalid rules.

## 7 Well-foundedness.

The most difficult part of a tableau proof is proving the well-foundedness of the $\sqsupset$ relations. Although the house-keeping and rule-checking provided by the tool is already useful, assistance with well-foundedness checking would be very welcome. This, even more so than the rest of the implementation, requires careful consideration of representation problems. It is quite possible to have a tableau and net that are easily represented in our tool, but whose well-foundedness proofs require reasoning power far beyond that available to any tool (or indeed, beyond that so far available to human mathematicians—see [Bra91b]). On the other hand, such pathological cases are, we hope, unlikely to occur in practical verification, where if a liveness property holds, it probably holds for some fairly simple reason. The theoretical question of finding classes of models for which a given reasoning power suffices, is of great interest, but also of considerable difficulty. Therefore, the well-foundedness checking assistance in the current implementation is designed to be both sufficient for many of our toy examples and relatively simple to implement; only experience will show how much more power is required in real applications.

The technique used in our prototype rests on the following easy result:

Assume the notation used in the definition of extended path. Suppose that we have a measure $\xi\colon \mathscr{S} \to X$ which maps each state to an element of a well-ordered set $X$. If the following condition holds, then $s@\mathbf{n} \dashrightarrow s'@\mathbf{n}'$ implies $\xi(s') < \xi(s)$ (and therefore $\sqsubset_{\mathbf{n}}$ is well-founded):

- if $\forall\, s, s'' \in \mathscr{S} \,.\, s@\mathbf{n} \longrightarrow s''@\mathbf{n}'' \;\Rightarrow\; \xi(s'') < \xi(s)$ then $\forall\, s'', s' \in \mathscr{S}, 1 \leq i \leq k \,.\, (s''@\mathbf{n}'' \dashrightarrow s'@\mathbf{n}_i \;\vee\; s''@\mathbf{n}'' \dashrightarrow s'@\mathbf{n}') \;\Rightarrow\; \xi(s') \leq \xi(s'')$; otherwise
- $\forall\, s, s'' \in \mathscr{S} \,.\, s@\mathbf{n} \longrightarrow s''@\mathbf{n}'' \;\Rightarrow\; \xi(s'') \leq \xi(s)$ and $\forall\, s'', s' \in \mathscr{S}, 1 \leq i \leq k \,.\, s''@\mathbf{n}'' \dashrightarrow s'@\mathbf{n}_i \;\Rightarrow\; \xi(s') \leq \xi(s'')$ and $\forall\, s'', s' \in \mathscr{S} \,.\, s''@\mathbf{n}'' \dashrightarrow s'@\mathbf{n}' \;\Rightarrow\; \xi(s') < \xi(s'')$.

That is, we are requiring that traversing the extended path decreases the measure, either by decreasing it along the top part and not increasing it in any subsidiary extended path, or by not increasing it in the top part or in any subsidiary loop and then decreasing it in the final part. The advantage is that we need only calculate with path relations: although the proposition refers to subsidiary extended paths, these can be checked by applying the proposition recursively (if $<$ is everywhere replaced by $\leq$, we have a result that deals with the $\leq$ clauses similarly). So the only calculation we need to do is to calculate a path relation $R$ and then determine whether $s \mathrel{R} s' \;\Rightarrow\; \xi(s') < \xi(s)$. The disadvantage is that we are requiring a much stronger property than the original tableau well-foundedness; it is possible, for example, for there to be a measure such that the subsidiary extended paths increase it, but the final path down to $\mathbf{n}'$ always compensates for this increases; this situation is not covered by this technique.

To implement this in our tool, we need a representation of binary relations on markings. Since we have a representation of sets (by linear (in)equalities), we represent relations in the same way, using primes to distinguish places in codomain markings from places in domain markings: e.g. $(\mathrm{A} = \mathrm{A}') \wedge (\mathrm{B}' = \mathrm{B} + 1)$ represents the relation $R$ such that $M \mathrel{R} M'$ iff $(M(\mathrm{A}) = M'(\mathrm{A})) \wedge (M'(B) = M(\mathrm{B}) + 1)$. Composition of such relations is handled by letting places with more than one dash represent existentially quantified variables, so that $(\mathrm{A}' = \mathrm{A}'' + 1) \wedge (\mathrm{A}'' = \mathrm{A} + 1)$ (which is the composition of $\mathrm{A}' = \mathrm{A} + 1$ with itself) should be read as $R$ such that $M \mathrel{R} M'$ iff $\exists M'' \,.\, (M'(\mathrm{A}) = M''(\mathrm{A}) + 1) \wedge M''(\mathrm{A}) = M(\mathrm{A}) + 1$. This interpretation of multiple primes does not require any work, since when such a relation is given to the emptiness checker, all variables are treated existentially anyway. It does mean that relations with multiple primes cannot be complemented; however, we never need to do so.

The calculation of path relations is straightforward: the individual steps are subrelations of the identity for non-modal rules, and are giving by the net firing rules and the choice functions for the modal rules. These steps are then composed to give the path relation.

There remains the question of how to represent the measure: this is done by representing not the measure itself, but rather the relation $R$ such that $M\,R\,M'$ iff $\xi(M) > \xi(M')$; it is the user's responsibility to determine that $R$ is such a relation, and then the tool checks the implications in the above proposition.

For an example, consider the net of Figure 1 and the 'finitely often' formula mentioned earlier: $\mu Y.\nu Z.[\mathrm{c}]Y \wedge [-\mathrm{c}]Z$. A suitable set for the root sequent is $\{\,\mathrm{A}+\mathrm{C}=1\,\}$.

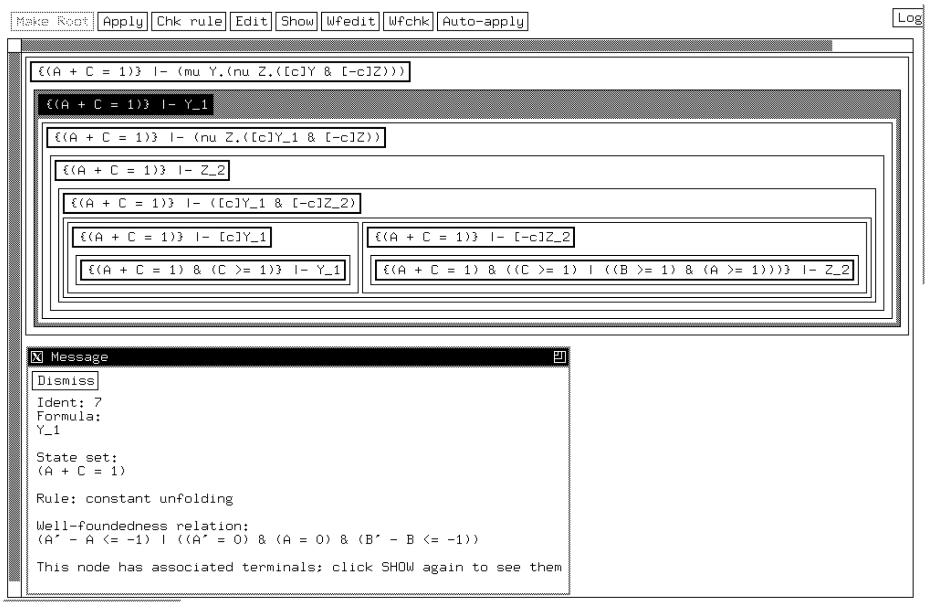


Figure 6

In Figure 6 the tableau for this has been developed to the point where all nodes are valid (using just one click of 'Auto-apply', since there are no $\vee$ or $\langle K\rangle$ rules). The grey background to the selected node shows that well-foundedness must be proved, and in fact a suitable relation has been typed in, as shown by the 'Show' message. The relation says that either A decreases or A remains at zero and B decreases—this corresponds to a measure $\xi$ mapping markings to the set $\omega \cdot 2$, with $\xi(M) = \omega + (M(\mathrm{A}) - 1)$ if $M(\mathrm{A}) > 0$, and $\xi(M) = M(\mathrm{B})$ otherwise. (Of course, in the net, $M(\mathrm{A})$ is never more than 1.)

Clicking on the 'Wfchk' button will perform all the implication checking required by the proposition, and the grey background turns black to indicate success; if any implication fails, a counter-example is presented just as for rule-checking.

## 8 Conclusions and further work.

We have described a window-based tool which assists in the construction of tableaux for proving temporal properties of arbitrary systems. The tableau management is independent of the model, and relieves the user of the burden of organizing tableau, keeping track of constants, etc. Model-dependent modules deal with representing infinite sets of states and calculating sets required by tableau rules, such as successor sets under certain transitions. Furthermore, the tool provides assistance in checking the complex well-foundedness relations in tableau, and indeed the user never sees these relations, but only has to provide a simpler relation. Here again there are both general and model-dependent components.

There is much scope for further work on this tool. Firstly, the interfaces between the different modules should be frozen and documented in order to move the tool out of the prototype stage. A reasoning module for CCS is a high priority, and will make the tool a useful companion to the Concurrency Workbench.

The representation of states for Petri nets needs further consideration; early experiments show that using linear inequalities to represent the whole state of non-trivial nets results in far too much work for the well-foundedness checker (owing mostly to the exponential blowup when already large relations are converted to disjunctive normal form). A suitable technique is probably to combine finite-state descriptions for parts of the net with infinite-state descriptions for others, and then to calculate successor sets etc. by simple exploration of the finite-state portion together with manipulation of the infinite-state representations.

Much work could be done on well-foundedness checking; this will be influenced both by theoretical investigation and by the results of experience.

The tool should be used in real cases; so far only toy examples have been used. The addition of a CCS module will make this easier, since the MPSS project already has several case studies to draw on.

## 9 Acknowledgements.

This work was supported by the SERC grants 'Mathematically Proven Safety Systems' (GR/F 38808) and 'Verification of Infinite State Systems—Theory, Method and Applications' (GR/H 16056).

'X Window System' is a trademark of the Massachusetts Institute of Technology.

'Maple' is a registered trademark of Waterloo Maple Software.

## 10 References.


[Bra91a] J. C. Bradfield, Proving temporal properties of Petri nets. *Advances in Petri Nets 1991* 29–47. LNCS 524 (1991).

[Bra91b] J. C. Bradfield, *Verifying Temporal Properties of Systems.* Birkhäuser, Boston, Mass. ISBN 0-8176-3625-0 (1991).

[BrS90] J. C. Bradfield and C. P. Stirling, Verifying temporal properties of processes. *Proc. CONCUR '90* 115–125. LNCS 458 (1990).

[BrS92] J. C. Bradfield and C. P. Stirling, Local model checking for infinite state spaces. To appear in *Theoret. Comput. Sci.* (1992).

[BrA91] G. Bruns and S. Anderson, The Formalization and Analysis of a Communications Protocol. In J. F. Lindeberg (ed.) *Proc. SAFECOMP '91* 7–12. IFAC Symposia Series (1991).

[ClE81] E. M. Clarke and E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic. LNCS 131 52–71 (1981).

[CES86] E. M. Clarke, E. A. Emerson and A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems* **8** 244–263 (1986).

[CPS89] R. Cleaveland, J. Parrow and B. Steffen, A semantics-based verification tool for finite state systems. *Proc. 9th IFIP Symp. on Protocol Specification, Testing and Verification.* North-Holland (1989).

[EmL86] E. A. Emerson and C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus. *Proc. First IEEE Symp. on Logic in Computer Science* 267–278 (1986).

[Koz83] D. Kozen, Results on the propositional mu-calculus. *Theoret. Comput. Sci.* **27** 333–354 (1983).

[KoP83] D. Kozen and R. Parikh, A decision procedure for the propositional mu-calculus. *Second Workshop on Logics of Programs* (1983).

[StW89] C. P. Stirling and D. J. Walker, Local model checking in the modal mu-calculus. *Proc. International Joint Conference on Theory and Practice of Software Development* 369–382. LNCS 351 (1989).

[StE89] R. S. Streett and E. A. Emerson, An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation* **81** 249–264 (1989).