Tree Automata as Infinite Countermodels for Propositional Logics

Zachary Ernst

Department of Philosophy
University of Missouri-Columbia

Automated Deduction and its Application to Mathematics – 2007
Outline

Motivation
  Easy Preliminaries
  The Need for Infinite Countermodels

Automata
  Finite Models, Finite Automata
  A New Kind of Tree Automata

Automating the Search

Questions
Outline

Motivation
- Easy Preliminaries
  - The Need for Infinite Countermodels

Automata
- Finite Models, Finite Automata
- A New Kind of Tree Automata

Automating the Search

Questions
Propositional Logics (in Polish)

- We’ll need only propositional logics with ‘→’ as the lone connective.
- The propositional letters ‘p, q, r, …’ are our variables.
- Easier to use Polish notation:
  - p → q is Cpq, and
  - (p → (q → r)) is CpCqr.
- The inference rules will be *modus ponens* and universal substitution.
  - From Cpq and p, infer q.
  - Uniformly replace variables in a derived formula with other formulae.
Logics and Theorems

- The set of theorems is a specified set of axioms, plus any formulae that can be derived from the axioms using those two rules of inference.
- A logic $\mathcal{L}$ is a set of axioms, plus rules of inference.
Matrix Countermodels

- Often, we need to show that a formula $\alpha$ is *not* a theorem of some propositional logic.
- The standard way to do so is by exhibiting a matrix model, like the ones generated by MACE4:

\[-P(c(x,y)) \mid P(y).\]
\[-P(c(x,y), c(y,z), c(z,x))).\]
\[-P(c(a,a)).\]
\[function(a, [0]),\]
\[function(c(_, _), [1, 2, 0, 3, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1, 3, 0]),\]
\[relation(P(_), [0, 0, 1, 0])\]

- The $P$ predicate represents True.
- The second clause says that every instance of the axiom must be true in the interpretation.
- The third clause asserts that the non-theorem is false in the model.
Outline

Motivation

Easy Preliminaries

The Need for Infinite Countermodels

Automata

Finite Models, Finite Automata

A New Kind of Tree Automata

Automating the Search

Questions
Logics Lacking the Finite Model Property

- But there are logics $\mathcal{L}$ in which:
  - Some formula $\alpha$ is not a theorem,
  - but every *finite* model validating the axioms of $\mathcal{L}$ and respecting $\mathcal{L}$’s rules of inference validates $\alpha$.

- These logics are said to lack the ‘finite model property’.

- So if you suspect that $\alpha$ is not a theorem of $\mathcal{L}$, but you can’t find a finite countermodel for it, the either:
  1. You haven’t tried hard enough...
  2. It really is a theorem (no countermodel exists)...
  3. Or it’s not a theorem, but it will take an infinite countermodel to show that.

- If you’re unlucky, then it’s option (3), and you’re up the creek.
An Easy Example

Consider the following easy logic $L_1$ with three axioms:

1. $Cpp$
2. $CCCcppq$
3. $CCCCcppcppaqqCCcppCqq$

To make things easier, let $Ix = Cxx$, so that we have:

1. $Ip$
2. $CCIpqp$
3. $CCIIpIq CIpIq$

(1) says that $Cpp$ is a theorem.

(2) says that if you’re ever able to prove an instance of $CCcpp$, then you can prove everything.

(3) says that if you’ve got $CI^n pq$, then you can prove $CI^{n-1} pq$. 
\( \mathcal{L}_1 \) Lacks the Finite Model Property

1. \( Ip \)

2. \( CCIpp \ q \)

3. \( CCIlpIq \ CIpIq \)

- It’s easy to show that every \textit{finite} model of \( \mathcal{L}_1 \) validates every \textit{formula} whatsoever.
  1. Let \( \mathcal{M} \) be a finite model of \( \mathcal{L}_1 \), and consider the sequence \( Ip, IIp, IIIp, \ldots \).
  2. Because \( \mathcal{M} \) is finite, there will be some \( n > m \) such that \( \mathcal{M} \) ‘thinks’ that \( I^n p = I^m p \).
  3. Since \( \mathcal{M} \) validates \( Cpp \), it validates \( C I^n p I^m p \).
  4. But then axiom (3) derives \( C I^{m+1} p I^m p \).
  5. So axiom (2) derives \( q \), which is arbitrary.
Outline

Motivation
Easy Preliminaries
The Need for Infinite Countermodels

Automata
Finite Models, Finite Automata
A New Kind of Tree Automata

Automating the Search

Questions
Overview of the Method

The guiding idea here is simple.

- Finite matrix models evaluate terms the same way finite tree automata evaluate trees.
- So for logics lacking the finite model property, we can use automata more powerful than finite tree automata to define countermodels more powerful than finite matrix models.
- The right kind of tree automata (UCTA) can be described in first-order logic.
- So we can use MACE4 and PROVER9 to fully automate the search for infinite countermodels.
Finite Tree Automata

A finite tree automata $T = \langle \Sigma, Q, Q_f, \sigma \rangle$ where:

1. $\Sigma$ is an input language.
2. $Q$ is a set of states.
3. $Q_f$ is a set of accept states.
4. $\sigma$ is a transition function.

- Tree automata move from the leaves of a tree up to the root, changing state according to the label of the node ($\in \Sigma$) and the transition function $\sigma$.
- If the automaton is in an accept state ($\in Q_f$) when it reaches the root node, then we say that it accepts the tree.
The Analogy

The analogy between finite matrix models and finite tree automata is obvious:

- Formulae of the logic may be thought of as trees.
- $\Sigma$ is like the vocabulary of the logic.
- $Q$ is like the domain of the matrix model.
- $Q_f$ is like the set of designated (True) elements of the matrix model.
- $\sigma$ is like the evaluation defined by the matrix.
The Disanalogy Between Automata and Models

- One possible problem: The vocabularies of logics are infinite, but the set of leaf symbols is always finite.

- But not to worry! Countermodels do not require that we model the entire logic – we need only model the number of variables occurring in the non-theorem (this is really easy to show).

- (The awful tyranny of condensed detachment)
Motivation for Unification Counters

Return to the example of $\mathcal{L}_1$:

- In the sequence $Ip, IIp, IIIp, \cdots$ a finite matrix model (automaton) cannot keep track of how many times the $I$ operation has been applied.
- That’s the fact that drives the proof that $\mathcal{L}_1$ lacks the finite model property.
- So we need an automaton that can keep track of this value.
Unification Counter Tree Automata

A formal definition is highly tedious. But the idea is simple:

- A UCTA has a finite set of states $Q$, with $Q_f \subseteq Q$.
- But there is also a special formula $\mathcal{F}$ and a counter $C$ that can take any integer value.
- At each node of the tree, the automaton checks whether the current subtree unifies with $\mathcal{F}$.
  1. If it does unify, then the counter is incremented.
  2. If not, then the counter is reset to zero.
- To determine the current state at each node, the automaton checks the counter values at each of the immediate children. For our examples, this yields a pair $\langle \ell, r \rangle$.
- There are three transition functions, depending on whether $\ell < r$, $\ell = r$, or $\ell > r$. 
Example of a UCTA

Consider the following UCTA:

- \( p \) is mapped onto state 0 with counter value 0.
- \( Q = \{0, 1, 2\} \), with \( Q_f = \{1, 2\} \).
- The unification formula \( F = Cxx \).
- The transition functions are defined as follows:

\[
\begin{align*}
\text{=} & \quad 0 & 1 & 2 \\
\text{>} & \quad 0 & 1 & 2 \\
\text{<} & \quad 0 & 1 & 2 \\
0 & [1 & 1 & 1] & 0 & [1 & 1 & 1] & 0 & [1 & 1 & 1] \\
1 & [0 & 1 & 1] & 1 & [0 & 0 & 1] & 1 & [0 & 2 & 2] \\
2 & [0 & 1 & 2] & 2 & [0 & 1 & 0] & 2 & [0 & 2 & 1]
\end{align*}
\]
Example of a UCTA Evaluation

- $p$ is mapped onto state 0 with $C = 0$.
- $Q = \{0, 1, 2\}$, with $Q_f = \{1, 2\}$.
- The unification formula $\mathcal{F} = Cxx$.

\[
\begin{array}{c|c|c|c}
= & 0 & 1 & 2 \\
\hline
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
2 & 0 & 1 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
> & 0 & 1 & 2 \\
\hline
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0 \\
\end{array}
\quad
\begin{array}{c|c|c|c}
< & 0 & 1 & 2 \\
\hline
0 & 1 & 1 & 1 \\
1 & 0 & 2 & 2 \\
2 & 0 & 2 & 1 \\
\end{array}
\]

Evaluate $CCppp$:

- $Cpp$ is evaluated by the ‘=’ matrix, so the state is 1. $Cpp$ unifies with $\mathcal{F}$, so the counter is 1.
- $CCppp$ is evaluated by the ‘>’ matrix, since the counter of the left subtree is 1 and the counter of the right subtree is 0. The state is therefore 0.
- But the formula $CCppp$ does not unify with $\mathcal{F}$, so the counter value is 0.
The Example (cont.)

That automaton has nice properties.

\[
= 0 \ 1 \ 2 \quad > \quad 0 \ 1 \ 2 \quad < \quad 0 \ 1 \ 2 \\
0[ \ 1 \ 1 \ 1 \ ] \quad 0[ \ 1 \ 1 \ 1 \ ] \quad 0[ \ 1 \ 1 \ 1 \ ] \\
1[ \ 0 \ 1 \ 1 \ ] \quad 1[ \ 0 \ 0 \ 1 \ ] \quad 1[ \ 0 \ 2 \ 2 \ ] \\
2[ \ 0 \ 1 \ 2 \ ] \quad 2[ \ 0 \ 1 \ 0 \ ] \quad 2[ \ 0 \ 2 \ 1 \ ]
\]

- It respects *modus ponens*.
- It validates all of the axioms of $\mathcal{L}_1$.
- Therefore, it validates every theorem of $\mathcal{L}_1$.
- But it doesn’t validate *everything*.

So the automaton really defines a model that is more powerful than any finite matrix model. (This model was found by computer)
Anderson’s System

John Anderson concocted the following three-axiom system, which we’ll call $\mathcal{L}_A$:

1. $Cpp$
2. $CCppq$
3. $CCCppqpCCCCppCppqr$

- It lacks the finite model property – any finite model of this system validates every formula whatsoever.
- But it is possible to find a UCTA that defines a more powerful model of $\mathcal{L}_A$...
The Automaton

The automaton has 108 transitions, but otherwise is defined as the earlier example:

```plaintext
function(e(_,_), [ 1, 1, 1, 1, 1, 1,  
  0, 4, 2, 0, 0, 5,  
  0, 4, 4, 0, 0, 0,  
  0, 0, 0, 5, 0, 0,  
  0, 5, 5, 5, 5, 0,  
  0, 5, 5, 0, 0, 5 ]),

function(g(_,_), [ 1, 1, 1, 1, 1, 1,  
  0, 2, 0, 0, 0, 5,  
  0, 0, 3, 0, 0, 3,  
  0, 0, 0, 2, 0, 1,  
  0, 0, 0, 0, 0, 0,  
  0, 0, 0, 0, 0, 0 ]),

function(l(_,_), [ 1, 1, 1, 1, 1, 1,  
  0, 1, 1, 1, 1, 1,  
  0, 1, 0, 0, 0, 0,  
  0, 0, 0, 0, 0, 0,  
  0, 3, 3, 0, 0, 3,  
  0, 2, 2, 0, 0, 2 ]),

relation(P(_), [ 0, 1, 1, 1, 1, 1 ])
```
Advantages of UCTA

In order to be useful in automated reasoning, any system for describing infinite countermodels must meet a few criteria:

1. The countermodels must be finitely presented.
2. They should be easy to enumerate.
3. It should be easy to tell whether they respect *modus ponens*.
4. And it should also be easy to tell whether they validate every instance of the axioms, and whether they invalidate the non-theorem.

UCTA meet all four conditions:

1. Presentation is easy – just like finite matrix models.
2. Enumeration is trivial.
3. The check for *modus ponens* is just like for finite matrix models.
4. Checking axiom instances is a little trickier, but not too bad...
Checking an Axiom Instance

Suppose you want to check the formula $\text{CCC}pppq$. Because the choice of transition matrix does not depend on the state (only on the counter values at the children), we have the following cases:

- Let $L, E, G$ represent evaluation by the $<, =, >$ matrices, respectively.
- Obviously, $Cpp$ will always be evaluated with the ‘$=$’ matrix. So we can write the formula as $\text{CCE}pppq$.
- The counter value of $Cpp$ will be greater than the counter value of $p$, so the formula can be written as $\text{CGE}pppq$.
- But $q$ could be anything at all, so we have to account for two cases (since the counter value of $\text{CC}ppp$ must be zero).
- So the set of rewrites is $\{LGEpppq, EGEpppq\}$.
\( \mathcal{L}_A \) Input File

Longer formulae may be translated into big sets of clauses. Here’s the translation for Anderson’s \( \mathcal{L}_A \) (in MACE4 notation):

\[
\begin{align*}
P(e(x,x)). \\
P(l(g(e(x,x),x),y)) & \quad P(e(g(e(x,x),x),y)). \quad -P(p) . \\
-P(l(x,y)) & \quad -P(x) \quad P(y) . \\
-P(e(x,y)) & \quad -P(x) \quad P(y) . \\
-P(g(x,y)) & \quad -P(x) \quad P(y) .
\end{align*}
\]

\[
\begin{align*}
P(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(e(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(l(e(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(e(e(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),l(l(e(e(x,x),e(x,x)),y),z))). \\
P(e(l(l(e(x,x),y),x),l(l(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),l(e(e(e(x,x),e(x,x)),y),z))). \\
P(e(l(l(e(x,x),y),x),l(e(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),l(e(e(e(x,x),e(x,x)),y),z))). \\
P(e(l(l(e(x,x),y),x),l(e(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(e(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))). \\
P(l(l(l(e(x,x),y),x),e(l(e(e(x,x),e(x,x)),y),z))).
\end{align*}
\]

...plus many more clauses like the last set.
Implementation Notes

- If a finite model finder, such as MACE4, successfully finds such a model, then that model represents the state transitions for the required automaton.
- Furthermore, the operation of the automaton can be described in first-order logic. So it is possible to program PROVER9 to verify that the automaton is correct.
- I have done this for several examples, including \( \mathcal{L}_A \).
- The proofs are long – the proof showing that the automaton validates the third axion of \( \mathcal{L}_A \) is 1,389 steps, with a depth of 510. It was found in 1,736 seconds.
- MACE4 works well for all examples I have tried, except \( \mathcal{L}_A \). But PARADOX gets it almost immediately.
Questions

- Are UCTA powerful enough to find countermodels for every logic lacking the finite model property?
- If the answer is ‘no’, then is there a natural set of extensions to UCTA that will do the trick?
- For any given logic $L$, and a non-theorem $\alpha$, let $L$ be the smallest set containing all theorems of $L$ containing only variables in $\alpha$, and respecting *modus ponens*. What are the possible complexity classes for any given $L$?
- Are there automata that solve open problems in (e.g.) equivalential calculus?
- Can we provide simple, easier to understand, countermodels using this technique for problems whose known solutions are extremely complex, such as the $P \rightarrow W$ problem?