From Model Checking to Proof Checking ... and Back

Kedar Namjoshi
Bell Labs

April 29, 2005
Abstraction $\circ$ Model Checking $= \text{Deductive Proof}$

MODEL CHECKING

$M \models \phi$

PROOF CHECKING

$M \vdash \phi$

Certifying Model Checker

Completeness

Proof Lifting
I. From Model Checking to Proof Checking

We show how to build a “certifying” model checker, one that generates a proof to justify its result.

Why bother?

- Proofs generalize counterexample traces for failure
- A proof is an *independently-checkable* certificate for success (think PCC for temporal properties)
- A proof is a convenient data structure for interactive exploration and incremental model checking
The CTL logic is built out of atomic propositions, boolean operators, and the *temporal* operators $EX(\phi)$ ("$\phi$ holds of some successor"), $E(\phi W \psi)$ ("$\phi$ unless $\psi$"), and $E(\phi U \psi)$ ("$\phi$ until $\psi$").

Some derived operators:
- $EF(\phi)$ ("$\phi$ is reachable") = $E(\text{true} U \phi)$
- $AX(\phi)$ ("all successors satisfy $\phi$") = $\neg EX(\neg \phi)$
- $AG(\phi)$ ("$\phi$ is invariant") = $\neg EF(\neg \phi)$
The basic CTL operators can be defined as fixpoints of $\text{EX}$-formulas.

- $\text{EF}(\phi) = (\min Z : \phi \lor \text{EX}(Z))$

- $\text{E}(\phi \text{W} \psi) = (\max Z : \psi \lor (\phi \land \text{EX}(Z)))$

Fixpoint formulas can be re-worked into a structurally simple notation: alternating automata.
Simple Alternating Automata (SAA)

Theorem 0. Every CTL formula can be represented by an SAA of proportional size.

(Euichi) acceptance set, \( F \), is empty.

This is just the parse graph of \( (\exists(Z) (\exists(Z) P \land \exists(Z) \min Z : P) \lor \exists(Z) P) \).

Theorem 0. \( EF (P) \) has a 3-state automaton, with initial state \( q_0 \) for propositions and \( EX \).

A SAA is just like an NFA, except that the transition function \( \delta \) maps a state to a boolean formula over atomic propositions.
An Automaton-based proof system

To show that a program $M$ with state set $S$ and transition relation $R$ satisfies an automaton property $(Q, \hat{q}, \delta, F)$ we need, for each automaton state $q$:

- An *invariance* predicate, $\phi_q \subseteq S$, and

- A partial *rank function*, $\rho_q : S \rightarrow \mathbb{N}$

Roughly speaking, the invariance assertions state that any (reachable) state of $M$ satisfying $q$ falls within the “safe” set $\phi_q$. The rank function marks the “distance” to reaching a Büchi state; it is re-set when the distance is 0.
Conditions for a valid Proof

- **Consistency:** $\rho_q$ is defined for every state in $\phi_q$

- **Initiality:** Every initial state of $M$ satisfies $\phi$̂

- **Safety and Progress:** Based on $\delta(q)$
  
  - $l$ (a literal): $\phi_q(s) \Rightarrow l(s)$, for all $s$.
  
  - $(\lor j : q_j)$: (similarly for $\land$)
    $\phi_q(s) \Rightarrow (\exists j : \phi_{q_j}(s) \land (\rho_{q_j}(s) <_q \rho_q(s)))$

  - $\text{EX}(r)$: (similarly for $\text{AX}$)
    $\phi_q(s) \Rightarrow (\exists t : sRt : \phi_r(t) \land (\rho_r(t) <_q \rho_q(s)))$

The relation $a <_q b = \text{if } q \notin F \text{ then } a < b \text{ else } true$

Progress and safety have to be checked together because of the $\text{EX}$ and $\lor$ operators.
Generating a Proof-I

**Key: model check with automata instead of CTL**

1. Turn CTL specification into a simple automaton
2. Form an AND-OR product graph of the program $M$ and automaton $A$
3. Check the canonical property: does Player I have a winning strategy?

$$W_I = \max Z; \min Y :$$

$$tt \lor$$

$$\left( \text{OR} \land (F \Rightarrow \text{EX}(Z)) \land (\neg F \Rightarrow \text{EX}(Y)) \right) \lor$$

$$\left( \text{AND} \land (F \Rightarrow \text{AX}(Z)) \land (\neg F \Rightarrow \text{AX}(Y)) \right) $$
Generating a Proof-II

Now set:

1. the invariant \( \phi_q \) to be \( \{ s : (s, q) \in W_I \} \)
2. the rank \( \rho_q(s) \) to the index of the earliest stage for \( Y \) where \( (s, q) \) is added, during the last \( Z \) iteration.

This works!

Theorem 1  The proof system is sound and (relatively) complete.
Problem: we do not know before-hand whether the check succeeds or fails.

Immediate Solution: Generate proofs after normal model checking. (this requires two runs of the model checker)

Better Solution? Exploit duality. If $W_I$ fails to hold of all initial states, then its dual, $W_{II}$, holds of some initial state. So keep approximations for both $Y$ and $Z$, and use whichever is appropriate at the end.
# A Simple Example

## 2-process, Atomic Bakery Protocol

```plaintext
\[
\text{var } st_1, st_2 : \{N, W, C\} \\
(* \text{N=“Non-critical”, W=“Waiting”, C=“Critical” } *) \\
\text{var } y_1, y_2 : \text{natural} \\
\text{initially } (st_1 = N) \land (y_1 = 0) \land (st_2 = N) \land (y_2 = 0)
\]

<table>
<thead>
<tr>
<th>Event</th>
<th>Condition</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait_1</td>
<td>( st_1 = N )</td>
<td>( \leftarrow) ( st_1, y_1 := W, y_2 + 1 )</td>
</tr>
<tr>
<td>enter_1</td>
<td>( st_1 = W \land (y_2 = 0 \lor y_1 \leq y_2) )</td>
<td>( \leftarrow) ( st_1 := C )</td>
</tr>
<tr>
<td>release_1</td>
<td>( st_1 = C )</td>
<td>( \leftarrow) ( st_1, y_1 := N, 0 )</td>
</tr>
<tr>
<td>wait_2</td>
<td>( st_2 = N )</td>
<td>( \leftarrow) ( st_2, y_2 := W, y_1 + 1 )</td>
</tr>
<tr>
<td>enter_2</td>
<td>( st_2 = W \land (y_1 = 0 \lor y_2 &lt; y_1) )</td>
<td>( \leftarrow) ( st_2 := C )</td>
</tr>
<tr>
<td>release_2</td>
<td>( st_2 = C )</td>
<td>( \leftarrow) ( st_2, y_2 := N, 0 )</td>
</tr>
</tbody>
</table>
```
The Abstracted Protocol

Abstraction: \( b_1 = (y_1 = 0); b_2 = (y_2 = 0); b_3 = (y_1 \leq y_2) \)

\[
\begin{align*}
\text{var } & st_1, st_2 : \{N, W, C\} \\
\text{var } & b_1, b_2, b_3 : \text{boolean} \\
\text{initially } & (st_1 = N) \land b_1 \land (st_2 = N) \land b_2 \land b_3 \\
\text{wait}_1 & \quad st_1 = N \leftrightarrow st_1, b_1, b_2, b_3 := W, false, b_2, false \\
\text{enter}_1 & \quad st_1 = W \land (b_2 \lor b_3) \leftrightarrow st_1, b_1, b_2, b_3 := C, b_1, b_2, b_3 \\
\text{release}_1 & \quad st_1 = C \leftrightarrow st_1, b_1, b_2, b_3 := N, true, b_2, true \\
\text{wait}_2 & \quad st_2 = N \leftrightarrow st_2, b_1, b_2, b_3 := W, b_1, false, true \\
\text{enter}_2 & \quad st_2 = W \land (b_1 \lor \neg b_3) \leftrightarrow st_2, b_1, b_2, b_3 := C, b_1, b_2, b_3 \\
\text{release}_2 & \quad st_2 = C \leftrightarrow st_2, b_1, b_2, b_3 := N, b_1, true, b_1
\end{align*}
\]
For the mutual exclusion property $\phi = AG(\neg(C_1 \land C_2))$, the invariants are just the set of reachable states.
Let $\xi$ be a simulation relation from $M$ to $\overline{M}$. A proof $(\phi, \rho)$ on $\overline{M}$ can be concretized to a proof $(\phi', \rho')$ on $M$ by letting

$$
\phi'_q(s) \equiv (\exists t : s \xi t \phi_q(t)), \text{ and }
\rho'_q(s) = (\min t : s \xi t \land \phi_q(t) \colon \rho_q(t))
$$

So:

$$
\phi'_q(st_1, st_2, y_1, y_2)
= \quad (\text{by definition})
(\exists b_1, b_2, b_3 : b_1 \equiv (y_1 = 0) \land b_2 \equiv (y_2 = 0) \land b_3 = (y_1 \leq y_2) \land
\phi_q(st_1, st_2, b_1, b_2, b_3))
= \quad (\text{simplifying})
\phi_q(st_1, st_2, (y_1 = 0), (y_2 = 0), (y_1 \leq y_2))
$$
It is possible to design a model checker which generates an independently checkable proof of its results. This can be done quite easily: COSPAN modification (experimental) about 200 lines of C. Generated proofs have several applications ... and perhaps some as-yet-unknown ones!
Abstraction ◦ Model Checking = Deductive Proof

MODEL CHECKING

\[ M \models \phi \]

Abstraction

PROOF CHECKING

\[ \overline{M} \models \phi \]

Completeness

Certifying Model Checker

\[ M \vdash \phi \]

Proof Lifting
II. Completeness of Verification via Abstraction
(joint work with Dennis Dams)

Given: Program $M$, property $\phi$; to check $M \models \phi$
Construct Abstraction: a finite program $\overline{M}$
Model Check: whether $\overline{M} \models \phi$

An Abstraction Framework specifies the precise relationship between $M$ and $\overline{M}$.

**Soundness:** for any $M, \phi$: if $\overline{M} \models \phi$, then $M \models \phi$

**Completeness:** for any $M, \phi$: if $M \models \phi$, there exists an abstraction $\overline{M}$ such that $\overline{M} \models \phi$
Summary of New Results

For properties expressed in branching time temporal logics (e.g., CTL, CTL*, or the $\mu$-calculus)

* **Negative:** Several well-studied abstraction frameworks are *incomplete*. Examples: bisimulation [Milner71], modal transition system refinement [Larsen-Thomsen88]. This holds even with enhancements such as *fairness* or *stuttering*.

* **Positive:** A simple extension of modal transition systems with new *focus* operations gives rise to a complete framework.

This is intimately connected to the representation of properties by finite tree automata.
Completeness and “Small Model” Theorems

**Small Model Theorem** [Hossley-Rackoff 72, Emerson85]: Any satisfiable property of the $\mu$-calculus has a finite model.

Why doesn’t this settle the question?
... because the small model need not abstract $M$.

Example:

$N$ is a small model for the property “there is a reachable $Q$-state”

But $N$ and $M$ are unrelated by, say, simulation or modal refinement.
A (Kripke) MTS is a transition system with

- **two** transition relations: *may* (over-approximate) and *must* (under-approximate) transitions, with \( \text{must} \subseteq \text{may} \)

- a **3-valued** \( \{\text{true}, \text{false}, \bot\} \) propositional valuation at states

For temporal logics, *existential path* modalities (e.g., \( \text{EX} \)) are interpreted over must-transitions; *universal path* modalities (e.g., \( \text{AX} \)) over may-transitions.

The outcome of model checking is also 3-valued.
Abstraction with MTS’s

If \( c \sqsubseteq a \) then:
- \( \forall c' : c \rightarrow c' \Rightarrow (\exists a' : a \xrightarrow{\text{may}} a' \land c' \sqsubseteq a') \)
- \( \forall a' : a \xrightarrow{\text{must}} a' \Rightarrow (\exists c' : c \rightarrow c' \land c' \sqsubseteq a') \)

Program M

integer x;
L1: \{x is even\}
L2: if (*)
    then x := x+2
    else x := x+4;
L3:

\{L2, even(x)\}
\{L3, even(x)\}  \{L3, div3(x)\}

\[\text{may transition}\]
\[\text{must transition}\]
Program M
L0: initially even(x)
L1: while (x > 0)  
do x := x-2 od;
L2: x := -1

Let $\phi = E(even(x)W(x < 0))$.

**Theorem 2** No finite MTS abstracts $M$ and satisfies $\phi$.

Proof by contradiction. The property holds for must-paths in $\overline{M}$; so either (i) $even(x)$ holds forever, or (ii) by finiteness, $x$ is negative within a bounded number of steps. The must-abstraction enforces these properties at every initial state of $M$, a contradiction!
Consequences and Variations

...
State-of-the-art for Completeness

* Model Abstraction: abstract the model, preserve the property

- $\mu$-calculus: *fair Focused Transition System abstraction*

* Game Abstraction: abstract the model-checking game, preserve the winning condition.

- $\mu$-calculus: fair alternating refinement+choice [Nami Joshi 2003]
The Need for Focus Operations

Transition $a \xrightarrow{\text{must}} b$ exists only if every $c : c \sqsubseteq a$ has a transition to a state abstracted by $b$.

This forces any abstract MTS for our example to be infinite. E.g., $L_1 : \text{even}(x) \not\rightarrow L_2 : (x < 0)$; so the source must be split; say to $L_1 : (x < 0), L_1 : (x \geq 0) \land \text{even}(x)$.

But again $L_1 : (x \geq 0) \land \text{even}(x) \not\rightarrow (x < 0)$.

Can one somehow relax the must-transition definition? (Such a relaxation must preserve soundness.)
An alternating automaton for $E(\text{even}(x) \land W(x < 0))$

During model checking, each automaton state is associated with a set of program states.

*Can an automaton be viewed as an abstract transition system?*
A *focus* step splits an abstract state into a set of more precise abstract states (case-splitting).

A Focused Transition System (FTS) is an MTS with focus and (dual) de-focus steps. For our example:

- $a_0 : \overline{L0, L1 : even(x)}, L2 : (x < 0)$
- $a_1 : L2 : (x < 0)$
- $a_2 : \overline{L0, L1 : even(x)}$
- $a_3 : \overline{L0, L1 : even(x)}$
- $a_4 : \overline{L0, L1 : even(x)}$

Note the similarity to the automaton — this is no accident.
Completeness via Automata

**Theorem 4** For any $M$ and any $\mu$-calculus property $\phi$, if $M \models \phi$, there is a finite FTS $\overline{M}$ such that $\overline{M}$ both abstracts $M$ and satisfies $\phi$.

The FTS $\overline{M}$ may be obtained by: (i) converting $\phi$ to a *finite* alternating tree automaton $A_\phi$, then (ii) converting $A_\phi$ to an FTS $\hat{A}_\phi$ (roughly) as follows.

- **AX-move** $\Rightarrow$ may transition
- **EX-move** $\Rightarrow$ must transition
- **∨-move** $\Rightarrow$ focus transition
- **∧-move** $\Rightarrow$ de-focus transition
- acceptance condition $\Rightarrow$ fairness condition
Notice that $\overline{M} = \hat{A}_\phi$ is independent of $M$! Thus, $\hat{A}_\phi$ is a maximal model for $\phi$

By results of [Emerson-Jutla 1991], this maximal model has size linear in the size of $\phi$.


Maximal models reduce model checking to simulation-checking.
May-Must abstraction *does not guarantee* the existence of finite abstractions for existential temporal properties.

The key to obtaining completeness seems to be a notion of $\epsilon$-state-splitting we call a *focus* step.

FTS’s are intimately connected to alternating tree automata. It turns out [Dams-Namjoshi, VMCAI 2005] that non-deterministic automata suffice. In effect: transition systems + fairness + *choice*

FTS’s also ensure more precision in must-abstractions. (Cf. [de Alfaro-Godefroid-Jagadeesan, LICS 2004])
To sum up

Model Checking and Proof Checking are closely linked, with Abstraction as the “glue”.
I. From Model Checking to Proof Checking

[Stevens-Stirling, TACAS 1998] *Practical Model-Checking Using Games*

[Namjoshi, CAV 2001] *Certifying Model Checkers*

[Peled-Zuck, SPIN 2001] *From Model Checking to a Temporal Proof*

[Peled-Pnueli-Zuck, FSTTCS 2001] *From Falsification to Verification*

[Clarke-Jha-Lu-Veith, LICS 2002] *Tree-like Counterexamples in Model Checking*

[Tan-Cleaveland, CAV 2002] *Evidence-Based Model Checking*


[Gurfinkel-Chechik, TACAS 2003] *Proof-like counterexamples*

[Namjoshi, VMCAI 2003] *Lifting Temporal Proofs through Abstractions*

[Namjoshi, CAV 2004] *An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking*
II. ... and Back

[Uribe, Thesis 2000] *Abstraction-Based Deductive-Algorithmic Verification of Reactive Systems*

[Kesten-Pnueli, Inf. Comp. 2000] *Verification by augmented finitary abstraction*

[Namjoshi, CAV 2003] *Branching-Time Abstraction*

[Dams-Namjoshi, LICS 2004] *The Existence of Finite Abstractions for Branching Time Model Checking*

[Dams-Namjoshi, VMCAI 2005] *Automata as Abstractions*
FTS’s and Disjunctive MTS’s [Larsen-Xinxin 1990]

DMTS’s introduced to guarantee a solution to CCS equations of the form \( \{C_i(X) = E_i\} \)

DMTS’s split a must-transition into cases: instead of \( a \xrightarrow{\text{must}} b \),
allow \( a \xrightarrow{\text{must}} \{B_0, B_1, \ldots\} \) where the \( B_i \) are sets of abstract states.


FTS’s are different in that one first splits state, then constructs ordinary must transitions.