COMP30172: Advanced Algorithms
Automata-based Verification - I

Howard Barringer

Room KB2.20: email: howard.barringer@manchester.ac.uk

March 2009
Supporting and Background Material

- Copies of key slides (already handed out)

- Chapter 9 of the Clarke, Grumberg and Peled “Model Checking”

- Two classic papers (hard):

See: http://www.cs.rice.edu/~vardi/papers for online versions of these papers
Supporting and Background Material

- Copies of key slides (already handed out)
- Chapter 9 of the Clarke, Grumberg and Peled “Model Checking”
- Two classic papers (hard):
See: http://www.cs.rice.edu/~vardi/papers for online versions of these papers
Supporting and Background Material

• Copies of key slides (already handed out)

• Chapter 9 of the Clarke, Grumberg and Peled “Model Checking”

• Two classic papers (hard):

See: http://www.cs.rice.edu/ vardi/papers for online versions of these papers
First Topic

Model Checking - a brief introduction
Great Success Story
Temporal logics
Programs as models
Reachability Analyses
In Summary
Outline

Model Checking - a brief introduction

Great Success Story

Temporal logics

Programs as models

Reachability Analyses

In Summary
What is Model Checking?

- Push button verification technology
What is Model Checking?

- Push button verification technology
- Establish whether formally expressed properties are true of a given model
What is Model Checking?

- Push button verification technology
- Establish whether formally expressed properties are true of a given model
- The model will represent an *abstraction* of all behaviours of a given system
What is Model Checking?

- Push button verification technology
- Establish whether formally expressed properties are true of a given model
- The model will represent an *abstraction* of all behaviours of a given system
- The properties can characterise both desired and undesirable behaviours
A Simple Example

Consider the model to be an abstraction of a network-based message switching system.

Nodes of the network are connected via communication channels.
Nodes contain routing control information, and route incoming messages towards their destination.

A rather desirable property might be:

\(a \text{ correctly addressed message input to the network will, in due course, reach its destination}\)

An undesirable property might be:

\(a \text{ singly input message is delivered many times to the correct address}\)

Clearly we want the first property to be true, and the second to be false, for all possible runs of the system!
Why MC is so successful?

- MC is pushbutton technology
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
- There are many quality tools freely available
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
- There are many quality tools freely available
- Users can almost apply formal methods without knowing it
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
- There are many quality tools freely available
- Users can almost apply formal methods without knowing it
- Good success stories in hardware verification circles
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
- There are many quality tools freely available
- Users can almost apply formal methods without knowing it
- Good success stories in hardware verification circles
- Growing use as a sophisticated debugger for software
Why MC is so successful?

- MC is pushbutton technology
- MC can provide useful counter examples when properties fail
- There are many quality tools freely available
- Users can almost apply formal methods without knowing it
- Good success stories in hardware verification circles
- Growing use as a sophisticated debugger for software
- ...
But ... significant limitations

• The problem being solved is computationally intractable
But ... significant limitations

- The problem being solved is computationally intractable
- Naive application most likely leads to resource exhaustion, both space and time
But ... significant limitations

- The problem being solved is computationally intractable
- Naive application most likely leads to resource exhaustion, both space and time
- In general restricted to finite state models
But ... significant limitations

- The problem being solved is computationally intractable
- Naive application most likely leads to resource exhaustion, both space and time
- In general restricted to finite state models
- Adding real-time and stochastic information possible but even further restricts scale of application
But ... significant limitations

- The problem being solved is computationally intractable
- Naive application most likely leads to resource exhaustion, both space and time
- In general restricted to finite state models
- Adding real-time and stochastic information possible but even further restricts scale of application
- Some of the property logics are too complex for typical users
Outline

Model Checking - a brief introduction
Great Success Story
Temporal logics
Programs as models
Reachability Analyses
In Summary
A typical property logic

- Temporal logic introduced to CS in 1977 by Pnueli for describing (concurrent) program properties
- TLs are a form of modal logic
- E.g., Linear temporal logic augments propositional logic with future time modalities:
  \[ \text{Always}, \text{Sometime}, \text{Next}, \text{Until} \quad \text{and past time equivalents.} \]

\[ \text{Always}(\text{sndto}(m, d) \Rightarrow \text{Sometime } \text{rcvfrom}(m, d)) \]

- express both safety and liveness properties
Linear Temporal Logic Models

Structures in which LTL formulas evaluated can be thought of as sequences of states, each state providing interpretation to the atomic propositions used.

Let $\sigma = s_0, s_1, s_2, \ldots, s_n, s_{n+1}, \ldots$. $\sigma$ is usually taken as of infinite length.

$\sigma, i \models p$ iff $p$ is true in state $s_i$

$\vdots$

$\sigma, i \models \text{Next } \phi$ iff $\sigma, i + 1 \models \phi$

$\sigma, i \models \text{Always } \phi$ iff $\forall j \geq i. \; \sigma, j \models \phi$

$\sigma, i \models \text{Sometime } \phi$ iff $\exists j \geq i. \; \sigma, j \models \phi$
A graph-like representation for models

• LTL has special finite model property
A graph-like representation for models

- LTL has special finite model property
- Any satisfiable formula has a model that can be represented finitely
A graph-like representation for models

- LTL has special finite model property
- Any satisfiable formula has a model that can be represented finitely
- the finite representation is a directed graph-like structure
A graph-like representation for models

- LTL has special **finite model property**
- Any satisfiable formula has a model that can be represented finitely
- the finite representation is a directed graph-like structure
- and is often known as a Kripke structure

\[ K = (S \text{ — set of states,} \\
R \text{ — a total binary relation over } S, \text{i.e. } \subseteq (S \times S), \\
I \text{ — an interpretation for atomic propositions,} \\
\text{ i.e. } I : (S \rightarrow 2^{AP}), \\
S_0 \text{ — a set of initial states, i.e. } S_0 \subseteq S) \]

where \( AP \) is the set of atomic propositions

A path in \( K \) is an infinite sequence of states, \( s_0, s_1, \ldots, s_i, \ldots \) s.t. for all \( i \), \( (s_i, s_{i+1}) \in R \).
Consider the formula

\[ p \land \neg q \land \text{Next Always}(q \land \neg p) \]

Its models can be represented as a two-state graph

The initial node \( s_0 \) has just \( p \) true, all subsequent reachable nodes, uniquely represented by \( s_1 \) have just \( q \) true.
Another Example Structure

The initial state set $S_0$ is \{s_0, s_1, s_2, s_3\}.
The structure contains the models for

\[ p \Rightarrow \text{Sometime } q \]
Of interest are \textbf{paths} that reach either state $s_1$ or state $s_3$. Those paths represent all models for $p \Rightarrow \textbf{Sometime } q$. We must re-visit this issue later.
Outline

Model Checking - a brief introduction

Great Success Story
Temporal logics

Programs as models
Reachability Analyses
In Summary
Logical modelling of (concurrent) programs - I

Let $CP$ be a concurrent program over variables $V$ ranging over finite data set $D$

A state of $CP$ can be represented by a mapping of values from $D$ to variables $V$

which, in turn, can be formulated as a logical formula

Let $V$ be the set of variables $\{x, y, z, pc\}$.
Let $D$ be the set of values $\{1, 2, 3, 4\}$.
A state $s$ is a valuation of variables to values, e.g.

$$[x \mapsto 1, \ y \mapsto 3, \ z \mapsto 2, \ pc \mapsto 4]$$

and can be represented by 1st order formula

$$x == 1 \land y == 3 \land z == 2 \land pc == 4$$
Logical modelling of (concurrent) programs - II

Thus let a 1st order formula $S_0$ over variables $V$ represent the set of initial states of $CP$.

Think of $V$ as denoting the present state variables of $CP$. Let $V'$ denote next state variables.

E.g. a formula $x' == x + 1$ represents a transition where the next value of $x$ is the current value plus 1.

Can write a 1st order formula $R$ over $V$ and $V'$ whose valuations can be viewed as a set of pairs of states, i.e. a transition relation.

\[(pc == 1) \Rightarrow ((x' == x + 1) \land (y' == y) \land (pc' == 2))\]
\[(pc == 2) \Rightarrow ((y' == y - 2) \land (x' == x) \land (pc' == 3))\]
\[(pc == 3) \land (y > 0) \Rightarrow ((pc' == 1) \land (y' == y) \land (x' == x))\]

Can model/specify a concurrent program, or aspects thereof, in this way.
From 1st order logic representation to Kripke structure

Let atomic propositions \( AP \) be the set of identities \( v \equiv d \), \( v \in V \) and \( d \in D \). A proposition \( v \equiv d \) is true in a state \( s \) if \( s(v) \equiv d \)

Build \( K = (S, R, I, S_0) \) as:

- set of states \( S \) is set of all valuations for \( V \);
From 1st order logic representation to Kripke structure

Let atomic propositions $AP$ be the set of identities $v == d$, $v \in V$ and $d \in D$. A proposition $v == d$ is true in a state $s$ if $s(v) == d$.

Build $K = (S, R, I, S_0)$ as:

- set of states $S$ is set of all valuations for $V$;
- initial states $S_0$ is the subset of $S$ that satisfy $S_0$;
From 1st order logic representation to Kripke structure

Let atomic propositions $AP$ be the set of identities $v == d$, $v \in V$ and $d \in D$. A proposition $v == d$ is true in a state $s$ if $s(v) == d$.

Build $K = (S, R, I, S_0)$ as:

- set of states $S$ is set of all valuations for $V$;
- initial states $S_0$ is the subset of $S$ that satisfy $S_0$;
- for states $s$, $s'$, $R(s, s')$ holds if $R$ is true for assignments to variables $v$ and $v'$ according to $s$ and $s'$, respectively;
From 1st order logic representation to Kripke structure

Let atomic propositions \( AP \) be the set of identities \( v == d, \ v \in V \) and \( d \in D \). A proposition \( v == d \) is true in a state \( s \) if \( s(v) == d \)

Build \( K = (S, R, I, S_0) \) as:

- set of states \( S \) is set of all valuations for \( V \);
- initial states \( S_0 \) is the subset of \( S \) that satisfy \( S_0 \);
- for states \( s, s' \), \( R(s, s') \) holds if \( R \) is true for assignments to variables \( v \) and \( v' \) according to \( s \) and \( s' \), respectively;
- The interpretation function \( I : S \mapsto 2^{AP} \) is defined so that \( I(s) \) is the subset of all atomic propositions true in \( s \);
From 1st order logic representation to Kripke structure

Let atomic propositions \( AP \) be the set of identities \( v == d, \ v \in V \) and \( d \in D \). A proposition \( v == d \) is true in a state \( s \) if \( s(v) == d \)

Build \( K = (S, R, I, S_0) \) as:

- set of states \( S \) is set of all valuations for \( V \);
- initial states \( S_0 \) is the subset of \( S \) that satisfy \( S_0 \);
- for states \( s, s' \), \( R(s, s') \) holds if \( R \) is true for assignments to variables \( v \) and \( v' \) according to \( s \) and \( s' \), respectively;
- The interpretation function \( I : S \rightarrow 2^{AP} \) is defined so that \( I(s) \) is the subset of all atomic propositions true in \( s \);
- The relation \( R \) is extended to be total, i.e. \( (s, s) \) is added to \( R \) for all states \( s \) that have no successor.
Example - I

The concurrent program:

while (y > 0)
{x = x + 1 || y = y - 2}

has (logical) operational semantics:

\[(pc == 0) \land (y > 0) \Rightarrow (((pc' == 1) \lor (pc' == 3)) \land (x' == x) \land (y' == y))\]
\[(pc == 1) \Rightarrow ((pc' == 2) \land (x' == x + 1) \land (y' == y))\]
\[(pc == 2) \Rightarrow ((pc' == 0) \land (y' == y - 2) \land (x' == x))\]
\[(pc == 3) \Rightarrow ((pc' == 4) \land (y' == y - 2) \land (x' == x))\]
\[(pc == 4) \Rightarrow ((pc' == 0) \land (x' == x + 1) \land (y' == y))\]
Example - II

Assume initial state $s_0 = [x \mapsto 10, y \mapsto 1]$, the Kripke structure

with interpretation

\[
\begin{align*}
I(s_0) &= \{pc == 0, x == 10, y == 1\} \\
I(s_1) &= \{pc == 1, x == 10, y == 1\} \\
I(s_2) &= \{pc == 2, x == 11, y == 1\} \\
I(s_3) &= \{pc == 0, x == 11, y == -1\} \\
I(s_4) &= \{pc == 3, x == 10, y == 1\} \\
I(s_5) &= \{pc == 4, x == 10, y == -1\}
\end{align*}
\]

is generated.
Commentary

- The same structure is being used for models of a property language and for programs
Commentary

- The same structure is being used for models of a property language and for programs
- Need ways to analyze the structures and also compare them
Commentary

- The same structure is being used for models of a property language and for programs
- Need ways to analyze the structures and also compare them
- E.g. determine whether all paths of the program Kripke structure are contained within the set of valid paths of the temporal property Kripke structure — this is model-checking.
• The same structure is being used for models of a property language and for programs
• Need ways to analyze the structures and also compare them
• E.g. determine whether all paths of the program Kripke structure are contained within the set of valid paths of the temporal property Kripke structure — this is model-checking.
• Or might perform specific reachability analyses on the program Kripke structure
Commentary

- The same structure is being used for models of a property language and for programs.
- Need ways to analyze the structures and also compare them.
- E.g. determine whether all paths of the program Kripke structure are contained within the set of valid paths of the temporal property Kripke structure — this is model-checking.
- Or might perform specific reachability analyses on the program Kripke structure:
  - characterise a set of “bad” or “unsafe” states — BAD.
Commentary

• The same structure is being used for models of a property language and for programs

• Need ways to analyze the structures and also compare them

• E.g. determine whether all paths of the program Kripke structure are contained within the set of valid paths of the temporal property Kripke structure — this is model-checking.

• Or might perform specific reachability analyses on the program Kripke structure
  • characterise a set of “bad” or “unsafe” states — BAD
  • check that no state in BAD is reachable from any initial state
Outline

Model Checking - a brief introduction
Great Success Story
Temporal logics
Programs as models
Reachability Analyses
In Summary
A first application of graph algorithms - I

forward reachability

Given a Kripke structure $K = (S, S_0, R, I)$ use depth-first search to determine whether a path exists from state $s_i \in S$ to state $s_j \in S$. E.g.

- all states $s \in S$ are forward reachable from $s_0$
- but $s_2$ is not reachable from $s_4$, etc.

Of interest is the set of reachable states from the initial set $S_0$. 
A forward reachability algorithm

boolean f_Reachable ( s, targetSet, visitedSet) =

    if (s is in targetSet) return true;
    if (s is in visitedSet) return false;
    else
        for each successor s' of s and while not found do
            found = f_Reachable(s’, targetSet, visitedSet U {s});
        return found;

f_Reachable(s₀, BAD, {}) returns true if any BAD state is reachable from the state s₀.
Reachable Sets of States

Alternatively, compute the set of reachable states from a given state.

Set $f_{\text{ReachableStates}}(\text{state}, \text{visitedStates}) =$

\[
\begin{align*}
    &\quad \text{if (state in visitedStates) return visitedStates; } \\
    &\quad \text{else } \\
    &\quad \quad \text{newVisitedStates = visitedStates U \{state\}; } \\
    &\quad \quad \text{for each successor state’ of state do } \\
    &\quad \quad \quad \text{newVisitedStates U= } \\
    &\quad \quad \quad \quad f_{\text{ReachableState}}(\text{state’}, \text{newVisitedStates}); \\
    &\quad \quad \text{return newVisitedStates; }
\end{align*}
\]

$f_{\text{ReachableStates}}(s_0, \{\})$ returns the set of states reachable from state $s_0$
Backwards Reachable Sets of States

Of course, can compute sets of states that reach a particular subset of states.

Set \( b_{\text{ReachableStates}}(\text{state}, \text{visitedStates}) = \)

\[
\begin{align*}
\text{if} & \ (\text{state in visitedStates}) \ \text{return} \ \text{visitedStates}; \\
\text{else} & \\
\text{newVisitedStates} & = \text{visitedStates} \cup \{\text{state}\}; \\
\text{for each predecessor state'} \ & \text{of state do} \\
\text{newVisitedStates} & = \text{newVisitedStates} \cup \\
& \quad b_{\text{ReachableState}}(\text{state'}, \text{newVisitedStates}); \\
\text{return} & \ \text{newVisitedStates}; \\
\end{align*}
\]

\( b_{\text{ReachableStates}}(s_f, \{\}) \) returns the set of states, each of which has a path to state \( s_f \)
Model Checking - a brief introduction

Great Success Story
Temporal logics
Programs as models
Reachability Analyses

In Summary
Summing up so far ...

- Can reduce verification of logical properties of program models to graph problems
- Explicit state model checking works in this sort of way
- Must understand the computational complexity, i.e. cost
- How many states in a single process program over ten 32-bit variables?
- Reachability is linear in size of graph structure
- For temporal logics, the problem is much harder - but we will consider this later
- Next, we consider a restricted form of model checking based on finite word automata