EE 144/244: Fundamental Algorithms for System Modeling, Analysis, and Optimization Fall 2014

State-Space Exploration

Stavros Tripakis
University of California, Berkeley

State-Space Exploration

- Goal: explore **state-space** of a system (typically a transition system).
  - E.g., **reachability analysis**: visit all states **reachable** from the initial states.
- For finite-state systems, it can be done exhaustively and fully automatically! (in principle)
- Basic method for solving the **model checking** problem.
  - Turing award 2007: Clarke, Emerson, Sifakis.
- Established practice in the industry (mainly hardware, but increasingly also software).

The Model Checking Problem

Does a given system M (the *implementation*, e.g., a state machine or a transition system) satisfy a given temporal logic formula ϕ (the *specification*, e.g., an LTL or CTL formula) ?

$$M \stackrel{?}{\models} \phi$$

Meaning:

- If ϕ is LTL: **all** execution traces of the system must satisfy ϕ.
- If ϕ is CTL: the initial state of the system must satisfy ϕ.

Invariants

Suppose ϕ is of the form

$$\mathsf{G}\psi \qquad \text{or} \qquad \mathsf{AG}\psi$$

where ψ is a propositional formula (boolean expression on atomic propositions).

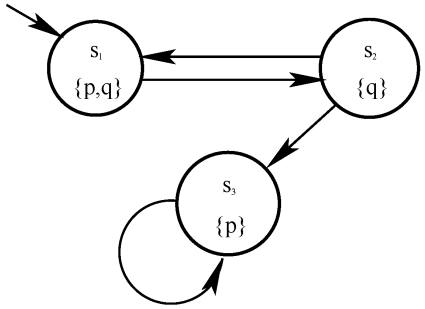
E.g.,

$$\mathsf{G}(p \vee q), \qquad \mathsf{G}(p \Rightarrow q), \qquad \cdots$$

Then ψ is called an **invariant**: it's a property that must hold at all **reachable** states.

Recall: Transition System (Kripke Structure)

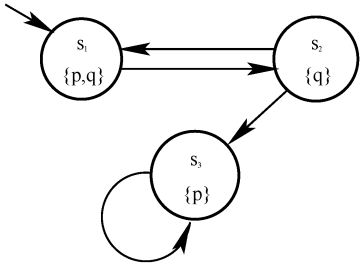
A tuple (P, S, S_0, L, R).



- P: set of atomic propositions, e.g., $P = \{p, q\}$.
- S: set of states, e.g., $S = \{s_1, s_2, s_3\}$.
- S_0: set of initial states, could be more than one, in this example just one: $S_0 = \{s_1\}$.
- $L : S \to 2^P$: labeling function, e.g., $L(s_1) = \{p, q\}$, $L(s_2) = \{q\}$, ...
- $R \subseteq S \times S$: transition relation, e.g., $R = \{(s_1, s_2), (s_2, s_1), (s_2, s_3), (s_3, s_3)\}$.

Reachable States

Given transition system (P, S, S_0, L, R).



A state $s \in S$ is called reachable if there exists a finite sequence of states

$$s_0, s_1, s_2, ..., s_k$$

such that:

1. $s_0 \in S_0$.
2. $\forall i = 0, ..., k-1 : (s_i, s_{i+1}) \in R$. We also write $s_i \to s_{i+1}$.
3. $s_k = s$.

Reachability Analysis

Visit all reachable states of a (typically finite) transition system.

At the same time, we can check whether every reachable state satisfies a given invariant ψ ...

... and therefore check that the system satisfies Gψ.

Caveat: Deadlocks

This assumes our system is **deadlock-free**, since only infinite paths count for the verification of Gψ.

Formally, s a deadlock state if $\nexists s' : s \rightarrow s'$.

How can we check that a given system is deadlock-free?

Use reachability analysis!

State-Space Exploration: Summary

- Reachability analysis: Check that system is never in an "incorrect" state, e.g.,
  - deadlock state
  - state which violates an invariant
  - e.g., "train is at intersection but gate is not lowered"
  - "autopilot is off but pilot thinks it is on"
  - ...
- Also the basis for checking *liveness* properties: every so often system does something useful.

State-Space Exploration Algorithms

- Enumerative (also called "explicit state").
  - These are basically search algorithms on directed graphs.
- Symbolic
  - Bounded model-checking using SAT/SMT solvers.
  - Symbolic reachability.

An Enumerative Algorithm: Depth-First Search

Assume given: Kripke structure (P, S, S_0, L, R).

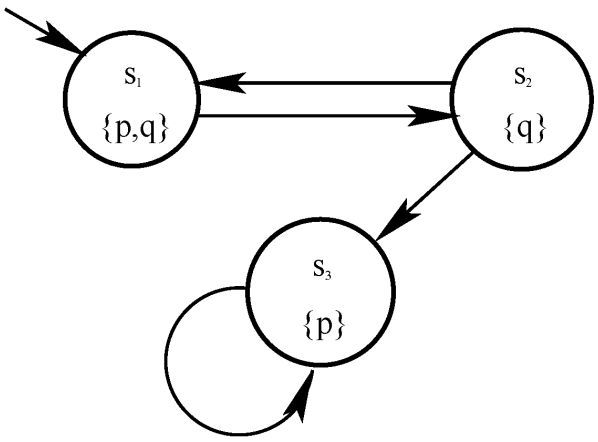
main:

1: $V := \emptyset$; /* V: set of visited states */
2: **for all** $s \in S_0$ **do**
3: DFS(s);
4: **end for**

DFS(s):

1: check s; /* is s a deadlock? is given $p \in L(s)$? ... */
2: $V := V \cup \{s\}$;
3: **for all** s' such that $(s, s') \in R$ **do**
4: **if** $s' \notin V$ **then**
5: DFS(s'); /* recursive call */
6: **end if**
7: **end for**

An Enumerative Algorithm: Depth-First Search



Let's simulate the algorithm on this graph.

An Enumerative Algorithm: Depth-First Search

Quiz:

- Does the algorithm terminate?
- Does it visit all reachable states?
- Does it visit any unreachable states?
- What is the complexity of the algorithm?

Enumerative Methods

Many algorithms: DFS, BFS, A*, ...

Many approaches to combat state-space explosion: partial-order reduction, symmetry reduction, bit-state hashing, ...

Lots of literature on the topic, including research papers [Godefroid and Wolper, 1991, Valmari, 1990, Holzmann, 1998] and textbooks [Clarke et al., 2000, Baier and Katoen, 2008].

In-depth discussion: Computer-Aided Verification course by Sanjit Seshia.

SYMBOLIC METHODS

Symbolic Methods: Why?

The plague of exhaustive verification: *state explosion*.

- A chip with 100 flip-flops: 2^{100} (potentially reachable) states.
- That is $1267650600228229401496703205376$ states.
- Even if each state costs 1 bit to store, this still makes $2^{100-60-8} = 2^{32} = 4,294,967,296$ exabytes ...
- Even if only $\frac{1}{32}$ states are reachable, this still makes $2^{100-5} = 2^{95}$ states.

Symbolic methods aim to improve this.

A seminal paper: "*Symbolic model checking:* 10^{20} *states and beyond.*" [Burch et al., 1990].

10^{20} is less than 2^{67}, but a great leap forward at that time.

Symbolic Representation of State Spaces

Key idea:

Instead of reasoning about individual states, reason about **sets** *of states.*

How do we represent a set of states?

Symbolic representation:

Set = predicate.

Set of states = predicate on state variables.

Symbolic Representation of Sets of States

Examples:

1. Assume 3 state variables, p, q, r, of type boolean.

$$S_1 : \quad p \vee q = \{p\overline{q}r, p\overline{q}\overline{r}, \overline{p}qr, \overline{p}q\overline{r}, pqr, pq\overline{r}\}$$

2. Assume 3 state variables, x, i, b, of types real, integer, boolean.

$$S_2 : \quad x > 0 \wedge (b \rightarrow i \geq 0)$$

How many states are in S_2?

Symbolic Representation of Transition Relations

Key idea:

> *Use a predicate on* **two copies** *of the state variables: unprimed (current state) + primed (next state).*

If $\vec{x}$ is the vector of state variables, then the transition relation R is a predicate on $\vec{x}$ and $\vec{x}'$:

$$R(\vec{x}, \vec{x}')$$

e.g., for three state variables, x, i, b:

$$R(x, i, b, x', i', b')$$

Symbolic Representation of Transition Relations

Examples:

1. Assume one state variable, p, of type boolean.

$$R_1 : \quad (p \rightarrow \neg p') \wedge (\neg p \rightarrow p')$$

   Which transition relation does this represent? Is it a relation or a function (deterministic)?

2. Assume one state variable, n, of type integer.

$$R_2 : \quad n' = n + 1 \vee n' = n$$

   Which transition relation does this represent? Is it a relation or a function (deterministic)?

Symbolic Representation of Kripke Structures

Kripke structure:

$$(P, S, S_0, L, R)$$

Symbolic representation:

$$(P, Init, Trans)$$

where

- $P = \{x_1, x_2, ..., x_n\}$: set of (boolean) state variables, also taken to be the atomic propositions.[1]
- Predicate $Init(\vec{x})$ on vector $\vec{x} = (x_1, ..., x_n)$ represents the set S_0 of initial states.
- Predicate $Trans(\vec{x}, \vec{x}')$ represents the transition relation R.

Basis of the language of NuSMV.

[1] this is done for simplicity, the two could be separated

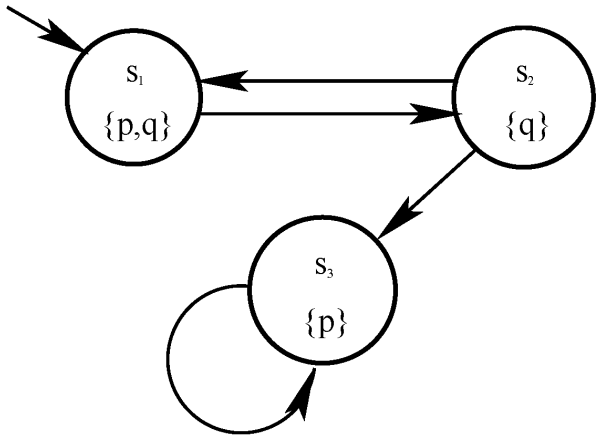
Example: NuSMV model

```
MODULE inverter(input)
VAR
  output : boolean;
INIT
  output = FALSE
TRANS
  next(output) = !input | next(output) = output
```

What is the Kripke structure defined by this NuSMV program?

What about P and L?

Example: Kripke Structure



Represent this symbolically.

Bibliography I

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Burch, J., Clarke, E., Dill, D., Hwang, L., and McMillan, K. (1990).
Symbolic model checking: 10^{20} states and beyond.
In *5th LICS*, pages 428–439. IEEE.

Clarke, E., Grumberg, O., and Peled, D. (2000).
Model Checking.
MIT Press.

Courcoubetis, C., Vardi, M., Wolper, P., and Yannakakis, M. (1992).
Memory efficient algorithms for the verification of temporal properties.
Formal Methods in System Design, 1:275–288.

Godefroid, P. and Wolper, P. (1991).
Using partial orders for the efficient verification of deadlock freedom and safety properties.
In *4th CAV*.

Holzmann, G. (1998).
An analysis of bitstate hashing.
In *Formal Methods in System Design*, pages 301–314. Chapman & Hall.

Huth, M. and Ryan, M. (2004).
Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press.

Bibliography II


Latvala, T., Biere, A., Heljanko, K., and Junttila, T. (2004).
Simple Bounded LTL Model Checking.
In *Formal Methods in Computer-Aided Design*, volume 3312 of *LNCS*, pages 186–200. Springer.


Robinson, J. (1965).
A machine-oriented logic based on the resolution principle.
Journal of the ACM, 12(1).


Valmari, A. (1990).
Stubborn sets for reduced state space generation.
LNCS 483.