
ELEC-E8110 Automation Systems Synthesis and Analysis

Igor Buzhinsky

igor.buzhinskii@aalto.fi

Aalto University
School of Electrical Engineering

2018
Symbolic model checking
Motivation

- State spaces can be very large

If there are ten 32-bit integer variables, how many states can the system have potentially?

\[2^{32} \approx 2^{10} \cdot 10^9\]

The so-called “state explosion” problem PC can probably handle (store in memory and process) only about \(10^9\) states...
State spaces can be very large

If there are ten 32-bit integer variables, how many states can the system have potentially?

\[ 2^{320} \approx 2^{10^9} \]

The so-called "state explosion" problem PC can probably handle (store in memory and process) only about \(10^9\) states...
State spaces can be very large
If there are ten 32-bit integer variables, how many states can the system have potentially? $2^{320} \approx 2.1 \cdot 10^{96}$
The so-called “state explosion” problem
Motivation

- State spaces can be very large
- If there are ten 32-bit integer variables, how many states can the system have potentially? \(2^{320} \approx 2.1 \cdot 10^{96}\)
- The so-called “state explosion” problem
- PC can probably handle (store in memory and process) only about \(10^9\) states...
State spaces can be very large

If there are ten 32-bit integer variables, how many states can the system have potentially? $2^{320} \approx 2.1 \cdot 10^{96}$

The so-called “state explosion” problem

PC can probably handle (store in memory and process) only about $10^9$ states...

Can we avoid explicit construction of the state graph?
State subsets as Boolean constraints (1)

Can you specify the set of reachable states as a Boolean formula?

\[
p \lor q
\]

What about only initial states?

\[
p \oplus q = p \land \neg q \lor \neg p \land q
\]
Can you specify the set of reachable states as a Boolean formula?

- $p \lor q$
Can you specify the set of reachable states as a Bohlean formula?

\( p \lor q \)

What about only initial states?
Can you specify the set of reachable states as a Boolean formula?

- \( p \lor q \)

What about only initial states?

- \( p \oplus q = p \land \neg q \lor \neg p \land q \)
State subsets as Boolean constraints (2)

What about the transition relation?

- $p, q$: values on this step
- $p', q'$: values on the next step

Quiz: specify the transition relation for the Kripke structure on the left as a Boolean formula:

$$(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p' \land q')$$

Alternative way: $$(p \land \neg q \land q' \land \neg p') \lor (q \land \neg p \land p' \land q') \lor (p \land q \land p')$$
State subsets as Boolean constraints (2)

- What about the transition relation?
- \( p, q \): values on this step
- \( p', q' \): values on the next step
- **Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

![Kripke structure diagram]

Alternative way: 

\[
(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p')
\]

\[
(p \land \neg q \land q' \land \neg p') \lor (q \land \neg p \land p' \land q') \lor (p \land q \land p')
\]
State subsets as Boolean constraints (2)

- What about the transition relation?
- \( p, q \): values on this step
- \( p', q' \): values on the next step
- **Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula
  
  \[ (p \land \neg q \rightarrow q' \land \neg p') \land \]

\[ \]
State subsets as Boolean constraints (2)

What about the transition relation?

$p, q$: values on this step

$p', q'$: values on the next step

**Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

$$(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land$$
State subsets as Boolean constraints (2)

What about the transition relation?

- \( p, q \): values on this step
- \( p', q' \): values on the next step

**Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

\[ (p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p') \]
State subsets as Boolean constraints (2)

What about the transition relation?

- $p, q$: values on this step
- $p', q'$: values on the next step

**Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

$$(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p')$$

Alternative way: $$(p \land \neg q \land q' \land \neg p')$$
State subsets as Boolean constraints (2)

- What about the transition relation?
- $p, q$: values on this step
- $p', q'$: values on the next step
- **Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

$$(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p')$$

Alternative way: $$(p \land \neg q \land q' \land \neg p') \lor$$
What about the transition relation?

- $p, q$: values on this step
- $p', q'$: values on the next step

**Quiz:** specify the transition relation for the Kripke structure on the left as a Boolean formula

$$(p \land \neg q \rightarrow q' \land \neg p') \land (q \land \neg p \rightarrow p' \land q') \land (p \land q \rightarrow p')$$

Alternative way: 

$$(p \land \neg q \land q' \land \neg p') \lor (q \land \neg p \land p' \land q') \lor (p \land q \land p')$$
Model checking with Boolean constraints?

- Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$
- Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$
- How to model-check $g = \mathbf{AG} \ h$, where $h$ is a Boolean formula?
Model checking with Boolean constraints?

- Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$
- Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$
- How to model-check $g = \text{AG} \: h$, where $h$ is a Boolean formula?
- Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps
Model checking with Boolean constraints?

- Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$
- Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$
- How to model-check $g = \textbf{AG} h$, where $h$ is a Boolean formula?
- Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps
- $f_0 := f_{\text{init}}$
Assume that our Kripke structure has atomic propositions $p_1, ..., p_n$

Boolean constraints $f_{\text{init}}[p_1, ..., p_n]$ and $f_{\text{trans}}[p_1, ..., p_n, p'_1, ..., p'_n]$

How to model-check $g = \mathbf{AG} h$, where $h$ is a Boolean formula?

Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps

$f_0 := f_{\text{init}}$; $f_i := f_{i-1} \lor \text{remove_primes}(\exists p_1, ..., p_n : f_{i-1} \land f_{\text{trans}})$
Model checking with Boolean constraints?

- Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$
- Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$
- How to model-check $g = \text{AG} \ h$, where $h$ is a Boolean formula?
- Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps
  - $f_0 := f_{\text{init}}$; $f_i := f_{i-1} \lor \text{remove_primes}(\exists p_1, \ldots, p_n : f_{i-1} \land f_{\text{trans}})$
  - If $f_i \land \neg h$ is satisfiable, then $g$ is false
Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$

Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$

How to model-check $g = \text{AG} \ h$, where $h$ is a Boolean formula?

Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps

- $f_0 := f_{\text{init}}$; $f_i := f_{i-1} \lor \text{remove}_\text{primes}(\exists p_1, \ldots, p_n : f_{i-1} \land f_{\text{trans}})$

If $f_i \land \neg h$ is satisfiable, then $g$ is false

If at some point $f_i$ and $f_{i-1}$ become equivalent, we can stop the procedure and conclude that $g$ is true
Model checking with Boolean constraints?

- Assume that our Kripke structure has atomic propositions $p_1, \ldots, p_n$
- Boolean constraints $f_{\text{init}}[p_1, \ldots, p_n]$ and $f_{\text{trans}}[p_1, \ldots, p_n, p'_1, \ldots, p'_n]$
- How to model-check $g = \mathbf{AG} h$, where $h$ is a Boolean formula?
- Compute a sequence of formulae $f_i$: the set of states reachable in $i$ steps
  
  $f_0 := f_{\text{init}}$; $f_i := f_{i-1} \lor \text{remove\_primes}(\exists p_1, \ldots, p_n : f_{i-1} \land f_{\text{trans}})$

- If $f_i \land \neg h$ is satisfiable, then $g$ is false
- If at some point $f_i$ and $f_{i-1}$ become equivalent, we can stop the procedure and conclude that $g$ is true

- How to perform all these symbolic operations efficiently? There are binary decision diagrams (BDDs), a reduced form of decision trees
Example of a BDD

- Solid arrows: variable is true
- Dashed arrows: variable is false
- If in the end we come to 1, then the formula is true for our assignment
- If we come to 0, it is false
Example of a BDD

- Solid arrows: variable is true
- Dashed arrows: variable is false
- If in the end we come to 1, then the formula is true for our assignment
- If we come to 0, it is false
- Which function is encoded in this BDD?
NuSMV model checker
NuSMV

- Open-source symbolic model checker
- Supports LTL and CTL
- Can be downloaded here: http://nusmv.fbk.eu/
- Command-line tool, models are specified in text files
- If an LTL specification is false, the corresponding counterexample can be visualized with the tool https://github.com/igor-buzhinsky/nusmv_counterexample_visualizer
MODULE main()
VAR
    p: boolean;
    q: boolean;
    c: 0..10;
INIT
    (c = 0) & (p)
TRANS
    (next(c) = (c + 1) mod 10) & (next(p) = !p)
CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))

Integers are supported
MODULE main()
VAR
  p: boolean;
  q: boolean;
  c: 0..10;
INIT
  (c = 0)
  & (p)
TRANS
  (next(c) = (c + 1) mod 10)
  & (next(p) = !p)
CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))

- Integers are supported
- Are the specifications in the end satisfied?
MODULE main()
VAR
  p: boolean;
  q: boolean;
  c: 0..10;
INIT
  (c = 0)
  & (p)
TRANS
  (next(c) = (c + 1) mod 10)
  & (next(p) = !p)

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))
MODULE main()
VAR
  p: boolean;
  q: boolean;
  c: 0..10;
INIT
  (c = 0)
  & (p)
TRANS
  (next(c) = (c + 1) mod 10)
  & (next(p) = !p)

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))

- Integers are supported
- Are the specifications in the end satisfied? – Yes
- What about q?
MODULE main()
VAR
   p: boolean;
   q: boolean;
   c: 0..10;
INIT
   (c = 0)
   & (p)
TRANS
   (next(c) = (c + 1) mod 10)
   & (next(p) = !p)

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))
NuSMV: alternative syntax

MODULE main()
VAR
    p: boolean;
    q: boolean;
    c: 0..10;
ASSIGN
    init(c) := 0;
    init(p) := TRUE;
    next(c) := c_plus_1 mod 10;
    next(p) := !p;
DEFINE
    c_plus_1 := c + 1;

CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))

- Explicit definitions for values changes
- Sub-expressions can be defined and reused
- Assignments can be nondeterministic, e.g.
  init(c) := \{0, 1\};
- INIT, TRANS, ASSIGN and DEFINE can co-exist

Use ASSIGN and DEFINE instead of INIT and TRANS where possible! It is easy to make a modeling error with INIT and TRANS.
MODULE main()
VAR
  p: boolean;
  q: boolean;
  c: 0..10;
ASSIGN
  init(c) := 0;
  init(p) := TRUE;
  next(c) := c_plus_1 mod 10;
  next(p) := !p;
DEFINE
  c_plus_1 := c + 1;
CTLSPEC AG(c != 10)
LTLSPEC G(p -> X(!p))

- Explicit definitions for values changes
- Sub-expressions can be defined and reused
- Assignments can be nondeterministic, e.g. init(c) := \{0, 1\};
- INIT, TRANS, ASSIGN and DEFINE can co-exist
- Use ASSIGN and DEFINE instead of INIT and TRANS where possible! It is easy to make a modeling error with INIT and TRANS
MODULE CYLINDER(fwd, back)

VAR
    pos: 0..5;

ASSIGN
    init(pos) := 0;
    next(pos) := fwd ? next_pos : back ? prev_pos : pos;

DEFINE
    next_pos := pos < 5 ? (pos + 1) : pos;
    prev_pos := pos > 0 ? (pos - 1) : pos;
    home := pos = 0;
    end := pos = 5;

- Modules can have inputs (in the declaration), and their variables and definitions can be interpreted as outputs
- C-style choice operator ?:
MODULE CONTROLLER(home, end)

VAR
    state: {moving_fwd, moving_back};

ASSIGN
    init(state) := moving_fwd;
    next(state) := case
        home: moving_fwd;
        end: moving_back;
        TRUE: state;
    esac;

DEFINE
    fwd := state = moving_fwd;
    back := state = moving_back;

- Example of explicit state machine modeling
MODULE main()
VAR
   -- this is the way to write comments, by the way
   cyl: CYLINDER(ctr.fwd, ctr.back);
   ctr: CONTROLLER(cyl.home, cyl.end);

LTLSPEC G F cyl.end -- TRUE
LTLSPEC G F cyl.home -- TRUE

- **Synchronous**: all the modules make a step together!
MODULE main()
VAR
   -- this is the way to write comments, by the way
   cyl: CYLINDER(ctr.fwd, ctr.back);
   ctr: CONTROLLER(cyl.home, cyl.end);

LTLSPEC G F cyl.end -- TRUE
LTLSPEC G F cyl.home -- TRUE

- **Synchronous**: all the modules make a step together!
- **How to model asynchronous interaction?**
SPIN model checker
Open-source **explicit-state** model checker

Supports LTL

Can be downloaded here: [http://spinroot.com/](http://spinroot.com/)

Can be run as a command-line tool, but also has GUI (iSpin)

Will not be covered in tutorials, assignments and the exam

You can try it yourself if you are interested
int pos = 0;
bool home = true, end, fwd, back;

// to be executed in a loop:
#define next_pos (pos < 5 -> (pos + 1) : pos)
#define prev_pos (pos > 0 -> (pos - 1) : pos)
pos = (fwd -> next_pos : (back -> prev_pos : pos));
home = pos == 0;
end = pos == 5;

- C-like syntax, but the choice operator has a different syntax
- C macros and other preprocessor directives are supported
- Conditional and loop statements (not shown) are very different, see online manuals if interested
bool home = true, end, fwd, back;
mtype = { moving_fwd, moving_back }; 
mtype state;

// to be executed in a loop:
state = (home -> moving_fwd :
  (end -> moving_back : state));
fwd = state == moving_fwd;
back = state == moving_back;

- mtype can be used for enumerations
int pos = 0;
bool home = true, end, fwd, back;
mtype = { moving_fwd, moving_back };  
mtype state;

init { do :: atomic {  // a loop of atomic steps
  // <plant loop code>
  // <controller loop code>
} od }

ltl visiting_end { [] <> end };  // G F end, true
ltl visiting_home { [] <> home }; // G F home, true

- Using this pattern, PLC-like applications can be modeled
Like UPPAAL, SPIN can verify asynchronous applications

- Multiple processes are supported
- `init` is executed in the beginning
- Other process types can be declared with the keyword `proctype`
- Their instances can be spawned with the keyword `run`
- Processes can execute asynchronously, unless explicitly constrained (e.g. by channels)
- *Partial order reduction* is used to reduce the state space in case of asynchrony


SPIN online references: [http://spinroot.com/spin/Man/](http://spinroot.com/spin/Man/)
User-friendly model checking
Why is it difficult to adopt model checking in industry?

- Efforts of formal modeling
- Human factor during modeling
- State space explosion: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space
- Model complexity can still be problematic for symbolic model checkers

Knowledge and experience are required to use formal methods correctly and efficiently.

How to mitigate this problem?
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling

- Human factor during modeling

- State space explosion: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space

- Model complexity can still be problematic for symbolic model checkers

Knowledge and experience are required to use formal methods correctly and efficiently

How to mitigate this problem?
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling
- **Human factor** during modeling
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling
- **Human factor** during modeling
- **State space explosion**: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling
- **Human factor** during modeling
- **State space explosion**: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space
- **Model complexity** can still be problematic for symbolic model checkers
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling
- **Human factor** during modeling
- **State space explosion**: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space
- **Model complexity** can still be problematic for symbolic model checkers

- **Knowledge and experience are required to use formal methods correctly and efficiently**
Why is it difficult to adopt model checking in industry?

- **Efforts** of formal modeling
- **Human factor** during modeling
- **State space explosion**: in explicit-state model checkers, verification time and required RAM generally grows linearly with the growth of the state space
- **Model complexity** can still be problematic for symbolic model checkers

- Knowledge and experience are required to use formal methods correctly and efficiently
- How to mitigate this problem?
Patterns by Dwyer et al. (1998, 1999): example 1

Absence

Intent
To describe a portion of a system’s execution that is free of certain events or states. Also known as Never.

Example Mappings

**CTL** $P$ is false:
- Globally: $AG(\neg P)$
- Before $R$: $A[\neg P U (R \lor AG(\neg R))]$
- After $Q$: $AG(Q \rightarrow AG(\neg P))$
- Between $Q$ and $R$: $AG(Q \rightarrow A[\neg P U (R \lor AG(\neg R))])$
- After $Q$ until $R$: $AG(Q \rightarrow \neg E[\neg R U (P \land \neg R)])$

**LTL** $P$ is false:
- Globally: $\Box(\neg P)$
- Before $R$: $\Diamond R \rightarrow \neg P U R$
- After $Q$: $\Box(Q \rightarrow \Box(\neg P))$
- Between $Q$ and $R$: $\Box((Q \land \Diamond R) \rightarrow (\neg P \land \Diamond(\neg P U R)))$
- After $Q$ until $R$: $\Box(Q \rightarrow (\neg P \land \Diamond(\neg P U (R \lor \Box \neg P))))$

- “A property specification pattern is a generalized description of a commonly occurring requirement on the permissible state/event sequences in a finite-state model of a system”
Response

Intent

To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect, within a defined portion of a system’s execution. Also known as Follows and Leads-to.

Example Mappings

In these mappings \( P \) is the cause and \( S \) is the effect.

**CTL** \( S \) responds to \( P \):

- Globally: \[ AG(P \rightarrow AF(S)) \]
- Before \( R \): \[ A[(P \rightarrow A[\neg R \cup ((S \land \neg R) \lor AG(\neg R)]) \cup (R \lor AG(\neg R)))] \]
- After \( Q \): \[ AG(Q \rightarrow AG(P \rightarrow AF(S))) \]
- Between \( Q \) and \( R \): \[ AG(Q \rightarrow A[(P \rightarrow A[\neg R \cup ((S \land \neg R) \lor AG(\neg R)]) \cup (R \lor AG(\neg R))]) \]
- After \( Q \) until \( R \): \[ AG(Q \rightarrow \neg E[\neg R \cup \neg(P \rightarrow A[\neg R \cup S]) \land \neg R]) \]

**LTL** \( S \) responds to \( P \):

- Globally: \[ \Box(P \rightarrow \Diamond S) \]
- Before \( R \): \[ (P \rightarrow (\neg R \cup (S \land \neg R))) \cup (R \lor \Box \neg R) \]
- After \( Q \): \[ \Box(Q \rightarrow \Box(P \rightarrow \Diamond S)) \]
- Between \( Q \) and \( R \): \[ \Box((Q \land \Diamond \Diamond R) \rightarrow (P \rightarrow (\neg R \cup (S \land \neg R))) \cup R) \]
- After \( Q \) until \( R \): \[ \Box(Q \rightarrow ((P \rightarrow (\neg R \cup (S \land \neg R))) \cup R) \lor \Box(P \rightarrow (\neg R \cup (S \land \neg R)))) \]
Patterns by Dwyer et al. (1998, 1999): hierarchy

- These patterns were extracted based on a volume of temporal properties collected from literature, student projects and other researchers.
- Note: different domains may have different prevailing patterns.
Visual specification languages (VSLs)

- Techniques to allow property representation and editing in a user-friendly, visual way
- Ideally, such techniques must be supported by tools
- Ideally, such tools must automatically translate visual specifications to textual formal specification languages (e.g. LTL, CTL)
- Unfortunately, this is not always so
