Introduction to Bounded Model Checking

Ramón Béjar Torres

Departament d'Informàtica
Universitat de Lleida

Bounded Model Checking - Introduction

Consider the following definition for a **finite state model** (FSM) representation of a system:

FSM

Is a 3-tuple $M = (S, S_0, R)$

- A finite set of states S
- A set of initial states $S_0 \subseteq S$
- A transition relation $R \subseteq S \times S$

What kind of systems can we model in this way ?

Bounded Model Checking - Introduction

Hardware:

- Combinatorial circuits and sequential circuits with finite memory
- Digital processors with finite memory

Software:

- Any program that works with a finite set of variables
- What about programs that work with dynamic memory ?
- What about programs that get input from files and other external sources ?

Bounded Model Checking - Introduction

What are the limits of the representation of digital systems by FSMs ?

Theorem 3.9.1 from Models Of Computation by John Savage

Any computation with T steps performed by a Turing Machine working with m b-bits memory cells can be simulated by a digital circuit with size $O(m\ b\ T)$ and depth $O(T \log(m\ b))$

So, in principle, given that *real* computers work always with finite memory (even when using dynamic memory) and with finite size files, there are no theoretical limitations in the representation of real digital systems

Moreover, a FSM can encode also unbounded (never ending) computations !

Bounded Model Checking - Introduction

What is so interesting about FSM representation of systems ?

There is a great amount of research about representation and checking of properties for FSMs

That is, properties that represent the good behaviour we want our FSM to satisfy

So, should the FSM satisfy certain specification related to its good behaviour, we write down the specification with a set of properties in certain **logical languages** and then check them with specialized algorithms

Extended FSMs or Kripke Structures

To build models with more rich information, that allow to associate more easily a FSM with concrete digital systems, we can use extended FSMs that label each state with a set of atomic propositions:

Kripke Structure

Is a 5-tuple $M = (S, S_0, R, AP, L)$

- A finite set of states S
- A set of initial states $S_0 \subseteq S$
- A transition relation $R \subseteq S \times S$
- AP is the set of atomic propositions
- $L : S \rightarrow 2^{AP}$ is a function that labels each state with the atomic propositions that are true in that state.

The labelling function gives specific **meaning** to the states.

Bounded Model Checking - Example 1

2-bit $[v_1 v_0]$ counter. Specification:

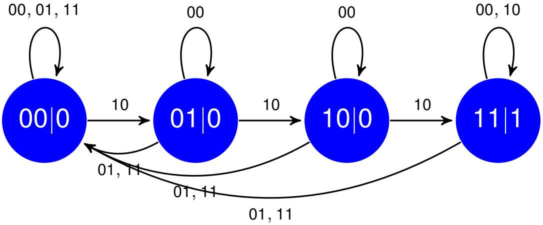
- 2 Inputs $[i_0 r]$: i_0 is the next value to accumulate, r signal to reset the acummulated value to 0 (ignore i_0)
- 1 Output: $[O]$: O will be 1 when accummulated value greater than [10]

Modeling it as a FSM/Kripke structure:

- State labellings: $v_1 v_0 | O$ (value of internal counter and of output signal)
- Identification of transitions: $i_0 r$ (value of inputs)

Bounded Model Checking - Example 1

We can model it as a FSM because no matter how long a sequence of inputs we feed into the counter, the set of possible states is finite !



Transitions labeled with all the possible inputs $i_0 r$ that produce such transition

Bounded Model Checking - Example 1

Some properties we may be interested in checking for this model:

- Starting from 00|0, **any sequence** of inputs reaches the state 11|1 at some point ?
- Starting from 00|0, **is there any sequence** of inputs that reaches the state 11|1 at some point ?

How would you check them with an algorithm ?

These properties, and many other more complex, can be expressed in different temporal logics and checked (decide if they are true in the FSM) with algorithms developed by Edmund M. Clarke and others

Bounded Model Checking - Example 2

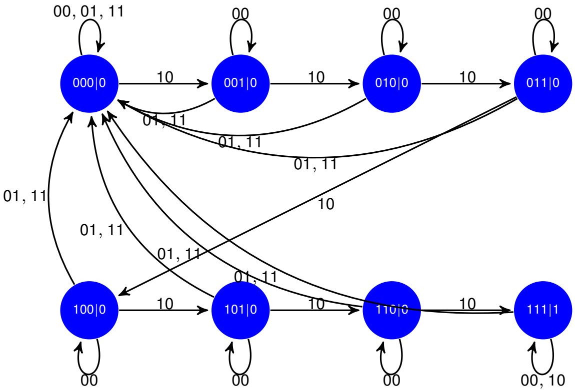
3-bit $[v_2 v_1 v_0]$ counter. Specification:

- 2 Inputs $[i_0 r]$: i_0 is the next value to accumulate, r signal to reset the acummulated value to 0 (ignore i_0)
- 1 Output: $[O]$: O will be 1 when accummulated value greater than [110]

Modeling it as a FSM:

- States: $v_2 v_1 v_0 | O$ (value of internal counter and of output signal)
- Transitions: $i_0 r$ (value of inputs)

Bounded Model Checking - Example 2



Increasing the size of the system by one bit doubles the state space size !

Bounded Model Checking - Example 3

Small program (two steps):

$$\begin{aligned} a &= 1 + i_0 \\ a &= a * i_1 \end{aligned}$$

with $i_0, i_1 \in \{1, 2, 3\}$

How to analyze it using a model based on a FSM ?

Bounded Model Checking - Example 3

Idea: Replace each update of a variable with a new copy of the variable

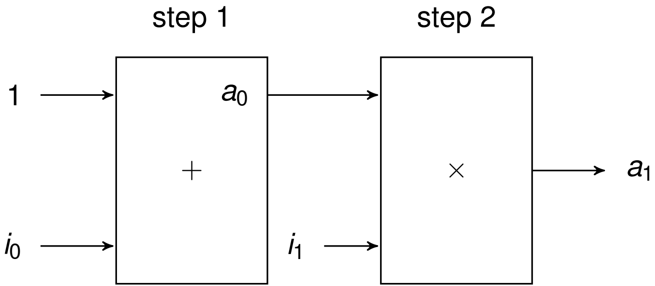
Renamed program (two steps):

$$\begin{aligned} a_0 &= 1 + i_0 \\ a_1 &= a_0 * i_1 \end{aligned}$$

with $i_0, i_1 \in \{1, 2, 3\}$

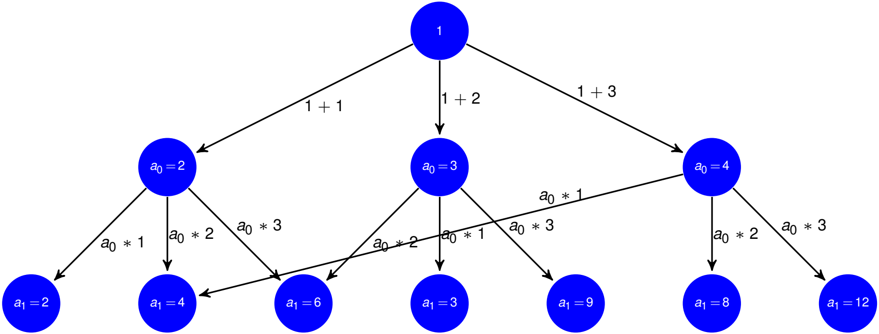
Bounded Model Checking - Example 3

That is, we view the program as a two-step sequential circuit:



So, we can model it as a FSM in a similar way as with *real* circuits

Bounded Model Checking - Example 3



Observe the growth of the number of states after each step

What can happen when we add one more step to the program ?

Bounded Model Checking - Example 4

One more step:

$$\begin{aligned} a &= 1 + i_0 \\ a &= a * i_1 \\ b &= a * i_2 \end{aligned}$$

with $i_0, i_1 \in \{1, 2, 3\}$ and $i_2 \in \{1, 2\}$

How does the corresponding FSM change ?

One more step:

$$\begin{aligned} a_0 &= 1 + i_0 \\ a_1 &= a_0 * i_1 \\ b_2 &= a_1 * i_2 \end{aligned}$$

with $i_0, i_1 \in \{1, 2, 3\}$ and $i_2 \in \{1, 2\}$

Bounded Model Checking - Example 4

One more step:

$$\begin{array}{rcl} a & = & 1 + i_0 \\ a & = & a * i_1 \\ b & = & a * i_2 \end{array}$$

with $i_0, i_1 \in \{1, 2, 3\}$ and $i_2 \in \{1, 2\}$

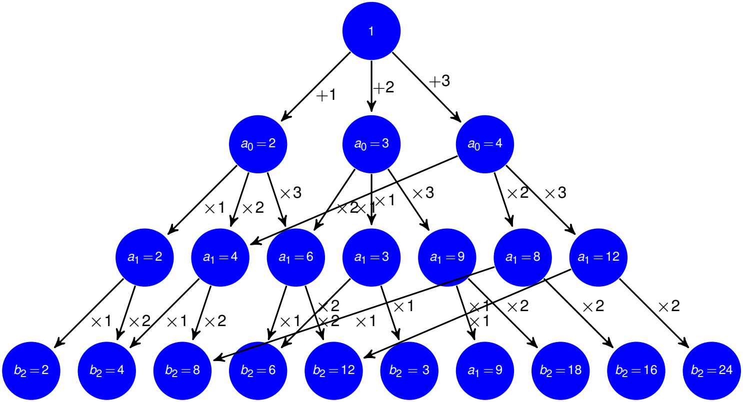
How does the corresponding FSM change ?

One more step:

$$\begin{array}{rcl} a_0 & = & 1 + i_0 \\ a_1 & = & a_0 * i_1 \\ b_2 & = & a_1 * i_2 \end{array}$$

with $i_0, i_1 \in \{1, 2, 3\}$ and $i_2 \in \{1, 2\}$

Bounded Model Checking - Example 4



Even for a bounded small number of transitions starting from a node, the size of the model can increase exponentially and be quite large for a small number of program steps !

Bounded Model Checking without Explicit Models

Although efficient algorithms (linear time) for checking any property expressed in quite expressive temporal logics, like CTL, exist, the problem is the **typical size** of FSMs for real systems

Instead of working with **explicit models** that can have exponential size with respect to some parameters of the model, as we have seen, another approach followed has been using implicit models.

That is, use programs/formulas where every assignment to its variables that is a solution represents a valid execution trace through the FSM, so we can explore any path trough the FSM for checking a particular property

Bounded Model Checking without Explicit Models

To use an implicit model, we represent **sequences of states** where states are represented trough variables associated with the different atomic propositions of AP.

For example, in our first example, we need three variables v_1, v_0, O per each state in an execution trace:

- Trace for input sequence 1 (no reset): $[v_1^1 = 0, v_0^1 = 1, O^1 = 0]$ (3*1 variables)
- Trace for input sequence 010 (no reset): $[v_1^1 = 0, v_0^1 = 0, O^1 = 0], [v_1^2 = 0, v_0^2 = 1, O^2 = 0], [v_1^3 = 0, v_0^3 = 1, O^3 = 0]$ (3*3 variables)
- Trace for input sequence 011 (no reset): $[v_1^1 = 0, v_0^1 = 0, O^1 = 0], [v_1^2 = 0, v_0^2 = 1, O^2 = 0], [v_1^3 = 1, v_0^3 = 0, O^3 = 0]$ (3*3 variables)

Bounded Model Checking without Explicit Models

In the previous example, the state labelling variables we have used are first-order logic variables rather than propositional variables

However, for finite state systems, any first order logic representation can be transformed to a propositional logic representation.

But observe that when we have infinite execution traces (like it can happen in this example), you need in principle an infinite number of variables to represent those traces ! (if execution length is not **bounded**)

Bounded Model Checking without Explicit Models

If this sounds strange to you, remember those AI algorithms, like DFS and A^*, that perform exploration of typically exponential size state spaces and that **do not necessarily** need to expand the whole state space

Well, for algorithms like A^* in the worst case the whole state space may be expanded ! But not for DFS !

A very successful approach for BMC has been the representation of FSMs together with the properties to check as SAT formulas, and then use high performing SAT solvers to check the properties

Bounded Model Checking without Explicit Models

Good, but why do we talk about **Bounded** Model Checking, and not simply Model Checking ?

In a real system, there can exist input sequences that give place to arbitrary long execution paths when checking some properties of the system

And specially in a **buggy** system, this can happen when the system gets trapped into a never ending loop (subsequence of states that the system traverses over and over without ending)

Bounded Model Checking without Explicit Models

So, consider these two situations:

- We have a system that halts for any possible input, so given a finite set of possible inputs, there is a finite bound on the execution path length for those inputs
- We have a system that does not halt for some inputs

Alan M. Turing showed that we cannot distinguish a priori whether an algorithm will stop for a given input, and so we cannot know whether an algorithm will need a bounded number of steps for giving an answer for any possible input

This holds for algorithms, and for digital state machine models that are equivalent to Turing Machines !

Bounded Model Checking without Explicit Models

What about halting Turing Machines ?

Even in the particular case where we know that the algorithm **stops for every input**, there is no way to decide if a certain function of the size of the input is a good upper bound on the number of needed computation steps for the algorithm !

Check this result, by Emanuale Viola, at http://cstheory.stackexchange.com

Bounded Model Checking without Explicit Models

This is why in general we consider the **Bounded** Model Checking Problem:

> *Given a digital system we want to analyze, consider only checking properties for execution paths with length upper bounded by some quantity k*

For certain systems, like combinatorial circuits and real time embedded systems, we know **by their design** that they have a bound on their execution path length for any input (and we know that they always halt !)

Bounded Model Checking without Explicit Models

But what if we want to analyze a system/program that may be buggy ? The BMC approach will only allow to check properties that hold up to certain execution length !

Given that state of affairs, in this course:

- We present an approach for BMC for a real programming language : `Ansi-C` based on the BMC tool `CBMC`
- For hardware verification we will give only a brief introduction trought the tool `EBMC`