Summary: Parameterized verification through view abstraction

Ashwin Abraham

IIT Bombay

February 2023

Table of Contents

Abstract

- This paper presents an efficient framework for automatic verification of systems with a parametric number of concurrent, communicating processes.
- For each value of the parameter, the system has a finite number of processes.
- The problem of automatic verification is to ensure that the system behaves *correctly* for every value of the parameter.
- We say that a system behaves *correctly*, if it cannot reach a predefined set of *bad* configuration from a predefined set of *initial* configurations.
- For simplicity, we model each process as a Finite State Machine, and assume all the processes are copies of each other.
- Here, we parametrize over the size of the system (the number of processes).

An overview of our strategy for automatic verification

- We extract a model for each process in the system.
- We extract an *overapproximation* of the model to give an abstract model
- We determine the initial states and bad states in this abstract model
- Finally, we check if the bad states can be obtained from the initial states in this abstract model

The Topology of a system

- The topology of a system refers to which processes can communicate with (ie inspect the state of) other processes.
- Some common topologies are:
  1. Linear Topology:
     Here the processes are ordered in an array and each process can distinguish the states to its left and those to its right. It can inspect the state of any of these processes.
  2. Ring Topology:
     The processes are ordered in a ring, and each process can only inspect the state of the process next to it.
  3. Tree Topology:
     The processes are arranged in a tree and each process can distinguish its parent process and child processes and inspect their states.
  4. Multiset Topology:
     Each process can distinguish every other process and inspect their states.

The Topology of a system

We can represent the topology of a system by a directed graph $G(V, E)$ where the vertices correspond to processes, and there is an edge from vertex A to vertex B if process A can inspect the state of process B.

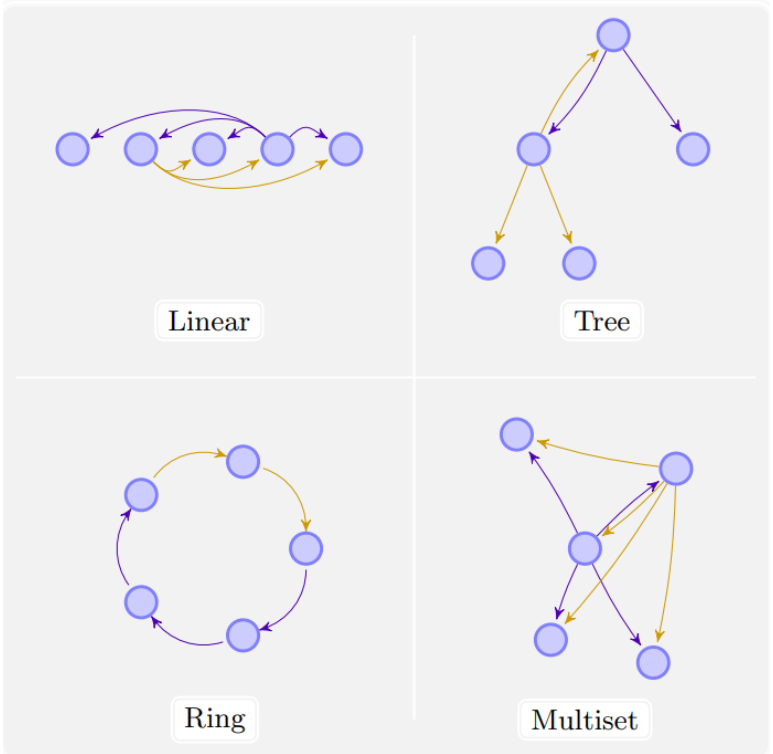

Figure: The graphs of the above mentioned topologies

Local and Global Transitions

- If a process transitions from one state to another without needing to inspect the states of other processes, the transition is known as a local transition.
- On the other hand, if the transition is dependent on the states of other processes, the transition is called a global transition.
- A major issue that arises is that global transitions are not checked atomically. The state of an inspected process may change while other processes are still being inspected.
- The method presented provides an elegant solution to this issue.

Other Transitions

- Broadcast Transitions: Here, an arbitrary number of processes change their state simultaneously. A broadcast transition has an initiator that causes a (possibly empty) subset of the processes that can inspect its state to transition.
- Rendezvous Transitions: Here a fixed number of processes change their state simultaneously.
- Shared Variable Update: Here, the shared variable is a part of the state of multiple processes, and when it is updated by any of the processes, the other processes change their state to reflect the updated value of the variable.
- Process Creation and Deletion make the topology of our system dynamic, but these can be reframed in terms of rendezvous transitions, as we will see later.

Formalism

To simplify our results, for now we will work with parametrized systems having identical processes and a linear topology, and that all transitions are either local or global. We assume for now that global checks are atomic, although we will soon drop this condition. We also assume that each process is governed by a finite state automaton.

- For a given topology, a parametrized system can be defined as the pair $\mathcal{P} = (\mathcal{Q}, \Delta)$, where $\mathcal{Q}$ is the set of *local states* of each process and Δ is a set of transition rules over processes.
- A local transition rule will be of the form $s_i \rightarrow s_i'$, where $s_i, s_i' \in \mathcal{Q}$ refer to the states of the i^{th} process before and after transition.
- A global rule is of the form $Qj \sim i : s_j \in S \implies s_i \rightarrow s_i'$. Here Q is a quantifier (either $\forall$ or $\exists$) and $\sim$ is some relation dependent on the topology of the system (for a linear topology it can be either $<$, $>$ or $\neq$) and S is some subset of $\mathcal{Q}$.

Formalism

- The configuration c of the system is a word over the alphabet $\mathcal{Q}$, ie, an array of states. We denote the set of all configurations as $\mathcal{C}$.
- $|c|$ denotes the number of processes. This is the parameter with which we parametrize the system.
- c_i denotes the state of the i^{th} process in configuration c, for $1 \leq i \leq |c|$.
- For a local rule $\delta \in \Delta$ such that $\delta = a \to b$, we say $c' = \delta(c, i)$ if c' is the result of applying δ on the i^{th} process. In particular, for $\delta(c, i)$ to be defined, c_i must be a, and $\delta(c, i)_i$ will be b.
- For a global rule $\delta \in \Delta$ such that $\delta = Qj \sim i : s_j \in S \implies a \to b$, $\delta(c, i)$ is defined iff $c_i = a$, and the condition $Qj \sim i : s_j \in S$ is satisfied. In this case, $\delta(c, i)_i = b$.
- We write $c \xrightarrow{\delta} c'$ if $c' = \delta(c, i)$ for some $i \in \{1 \dots c\}$, and we define $\xrightarrow{*}$ as the reflexive and transitive closure of $\xrightarrow{\delta}$ (possibly with different δs).

Reachability

- We say that a configuration c' is reachable from c if $c \xrightarrow{*} c'$.
- The *reachability problem* is defined by a parametrized system $\mathcal{P} = (\mathcal{Q}, \Delta)$, a set of initial states $\mathcal{I} \subseteq \mathcal{C}(\mathcal{P})$ and a set of bad states $\mathcal{B} \subseteq \mathcal{C}(\mathcal{P})$.
- We define the set $\mathcal{R}$ of reachable states as $\{c \in C : \exists c' \in \mathcal{I}, c' \xrightarrow{*} c\}$. We say a $\mathcal{P}$ is *safe* with respect to $\mathcal{I}$, $\mathcal{B}$ iff $\mathcal{R} \cap \mathcal{B} = \emptyset$.
- In order to define the *bad* set $\mathcal{B}$ independently of the parameter, we define it as the closure of a set of minimal bad configurations, ie $\mathcal{B} = \{c \in \mathcal{C} : \exists b \in B_{min} b \sqsubseteq c\}$, where $\sqsubseteq$ denotes a relation (known as the *covering* relation) dependent on the topology of the system (for a linear topology it'll be the subword relation).
- $\mathcal{I}$ and $\mathcal{R}$ are also defined independently of the parameter this way.
- $\mathcal{S}_k$ denotes $\{c \in S : |c| = k\}$ for $S = \mathcal{I}$, $\mathcal{B}$, $\mathcal{R}, \mathcal{C}$.

Szymanski's Protocol

- Szymanski's Protocol is a protocol that is used to ensure mutually exclusive access to a resource for a finite number of processes.
- It is based on the analogy of a waiting room with an entry door and an exit door that gives access to the resource.
- Each process has a flag with 5 possible values:
  - **0**: The process declares that it doesn't want to access the resource right now (it is outside the waiting room)
  - **1**: The process declares that it wants to enter the critical section, and it is outside the waiting room.
  - **2**: The process is waiting for other processes to enter the waiting room.
  - **3**: The process has just entered the waiting room.
  - **4**: The process is about to start or in the critical section.
- The flag of a process can be read by every other process but can be changed only by itself.

Szymanski's Protocol

- In the beginning, all the processes that request entry at roughly the same time, enter the waiting room one by one.
- The last of these closes the entry door and waits for the processes of lower IDs to leave the waiting room and finish accessing the resources.
- Once the lower ID processes are done, they change their flags to **0** or **1** to indicate this. The last process finishing resource access causes the entry door to be opened.
- For now, we assume that all these checks can be done atomically.

Szymanski's Protocol

The pseudocode for this protocol is as follows:

```
0  flag[i] := 1
1  wait until ∀j≠i : flag[j] ∈ {0, 1, 2}
2  flag[i] := 3
3  if ∃j≠i : flag[j] = 1 then
4        flag[i] := 2
5        wait until ∃j≠i : flag[j] = 4
6  end
7  flag[i] := 4
8  wait until ∀j < i : flag[j] ∈ {0, 1}
9  /* Critical Section */
10 wait until ∀j > i : flag[j] ∈ {0, 1, 4}
11 flag[i] := 0; goto 0;
```

Szymanski's Protocol

- The configuration of the system here is a word over the alphabet $\{0, 1, \cdots 11\}$.
- The initial configuration of the system has all processes in state 0.
- $\mathcal{I} = \{0, 0.0, 0.0.0 \cdots\}$
- The bad configurations are those where more than one process is in state 9 or state 10.
- $\mathcal{B} = \{9.9, 9.10, 10.9, 10.10, \cdots\}$

Verification

- The key insight here is that small instances of the system give enough information to predict the behaviour for all values of the parameter.
- On a high level, we can say that this occurs because the patterns that occur for lower values of the parameter repeat themselves for larger values.
- The configurations for higher parameters cover those of lower parameters, in a sense.
- Our algorithm automatically detects a cutoff point n such that we only need to compute $\mathcal{R}_1, \mathcal{R}_2 \cdots \mathcal{R}_n$ to determine whether $\mathcal{R} \cap \mathcal{B} = \emptyset$.
- To do this, we perform *overapproximation*, where we abstract the system such that $R \subseteq R'$. Now if we are able to show $R' \cap B = \emptyset$, we can conclude that $R \cap B = \emptyset$.

View Abstraction

- We first focus only on *atomically* checked global conditions. As mentioned earlier, we will remove this restriction soon.
- In *view abstraction*, we consider the configurations from the perspective of a few *fixed* number of processes. This abstraction is parametrized by k, the number of processes.
- The new construct made by considering the perspective of only the k processes taken is called a *view*.
- We have a certain freedom in choosing what information to discard and what to retain while constructing the view.
- For example, a simple example of a view would be a subword of the configuration c, with size k.

The Subword View

- Here, a view of a configuration c with size k is a subword of c with size k.
- We denote the set of views by $\mathcal{V}$ and the set of views of size *upto* k by $\mathcal{V}_k$. Note that a view is also a valid configuration, and therefore $\mathcal{V}_k \subseteq \mathcal{C}$.
- We now define two functions:
  1. The *abstraction* function $\alpha_k : \mathcal{C} \to 2^{\mathcal{V}_k}$ that maps a configuration to the set of all it's subwords of size at most k.

$$\alpha_k(c) = \{v \in \mathcal{V}_k : v \sqsubseteq c\}, \forall c \in \mathcal{C}$$

  2. We also define the *abstraction* for subsets of $\mathcal{C}$ as $\alpha_k : 2^{\mathcal{C}} \to 2^{\mathcal{V}_k}$ as

$$\alpha_k(S) = \{v \in \mathcal{V}_k : \exists c \in S : v \sqsubseteq c\} = \bigcup_{c \in S} \alpha_k(c), \forall S \subseteq \mathcal{C}$$

  3. The *concretization* function $\gamma_k : 2^{\mathcal{V}_k} \to 2^{\mathcal{C}}$ that maps a set of views to the set of configurations that can be reconstructed from those views, ie

$$\gamma_k(V) = \{c \in \mathcal{C} : \alpha_k(c) \subseteq V\}, \forall V \subseteq \mathcal{V}_k$$

Galois Connection

- We can order $2^{\mathcal{C}}$ and $2^{\mathcal{V}_k}$ by defining $x \leq y$ when $x \subseteq y$, $\forall x, y \in 2^{\mathcal{C}}$ or $\forall x, y \in 2^{\mathcal{V}_k}$.
- With this ordering, α_k and γ_k become monotonic functions.
- $\forall V \subseteq \mathcal{V}_k, \alpha_k(\gamma_k(V)) \subseteq V$. This is because $\forall c \in \gamma_k(V)$, $\alpha_k(c) \subseteq V$ (by the definition of γ_k).
- $\forall S \subseteq C, S \subseteq \gamma_k(\alpha_k(S))$. This is because $c \in S \implies \alpha_k(c) \subseteq \alpha_k(S)$ (by monotonicity), which implies that $c \in \gamma_k(\alpha_k(S))$.

If $(A, \leq)$ and $(B, \leq)$ are two posets, we say two monotone functions $F : A \to B$ and $G : B \to A$ form a Galois Connection iff

$$\forall a \in A, \forall b \in B, F(a) \leq b \Leftrightarrow a \leq G(b)$$

Now, $\alpha_k(A) \subseteq B \implies \gamma_k(\alpha_k(A)) \subseteq \gamma_k(B) \implies A \subseteq \gamma_k(B)$. Similiarly, we can prove the converse. Therefore, α_k and γ_k form a Galois connection.

Monotonicity of $\gamma_k \alpha_k$

Lemma:

$$\forall X \subseteq \mathcal{C}, \gamma_1(\alpha_1(X)) \subseteq \gamma_2(\alpha_2(X)) \cdots \subseteq X$$

Proof: