Model checking pushdown systems

R. Ramanujam

Institute of Mathematical Sciences, Chennai

jam@imsc.res.in

Sources of unboundedness

- Data manipulation: integers, lists, trees, pointers

Sources of unboundedness

- Data manipulation: integers, lists, trees, pointers
- Control structures: procedures, process creation

Sources of unboundedness

- Data manipulation: integers, lists, trees, pointers
- Control structures: procedures, process creation
- Asynchronous communication: unbounded FIFO queues (buffers)

Sources of unboundedness

- Data manipulation: integers, lists, trees, pointers
- Control structures: procedures, process creation
- Asynchronous communication: unbounded FIFO queues (buffers)
- Parameters: number of processes, delay duration

Sources of unboundedness

- Data manipulation: integers, lists, trees, pointers
- Control structures: procedures, process creation
- Asynchronous communication: unbounded FIFO queues (buffers)
- Parameters: number of processes, delay duration
- Real time: discrete, dense domains

Extended automata

- A generic way of modelling such systems is by finite state automata with guarded transitions.
- An extended automaton is equipped with a finite set of variables $X = \{x_1, \ldots, x_n\}$ with variable x_i taking values in set V_i.
- We have a finite set of guards G: each guard is a preicate over X.
- With each transition is associated an action, which is possibly a nondeterministic assignment to X.

Extended automata: semantics

A configuration is a tuple $(q, v_1, \ldots, v_n)$ where q is a state and v_i is a valuation for x_i.
The transition system of the extended automaton is over configurations:
$(q, v_1, \ldots, v_n) \Rightarrow (q', v_1', \ldots, v_n')$ if the automaton has a transition $q \xrightarrow{g,a} q'$, the values v_i satisfy guard g and the tuple $(v_1', \ldots, v_n')$ is a possible result of applying a to $(v_1, \ldots, v_n)$.

Some classes of extended automata

- Timed automata: Variables – clocks; guards – comparisons; actions: reset.

Some classes of extended automata

- Timed automata: Variables – clocks; guards – comparisons; actions: reset.
- Petri nets: Variables – counters; guards – $x = 0$; actions: * / - .

Some classes of extended automata

- Timed automata: Variables – clocks; guards – comparisons; actions: reset.
- Petri nets: Variables – counters; guards – $x = 0$; actions: * / - .
- FIFO automata: Variables – queues; guards – emptiness check; actions: insertion / deletion.

Some classes of extended automata

- Timed automata: Variables – clocks; guards – comparisons; actions: reset.
- Petri nets: Variables – counters; guards – $x = 0$; actions: * / - .
- FIFO automata: Variables – queues; guards – emptiness check; actions: insertion / deletion.
- Pushdown systems: Variables – stack; guards – emptiness check; actions: push / pop.

Reachability problem

- Given: An extended automaton E, a set I of initial configurations and a set D of dangerous configurations.
- Decide if some $d \in D$ is reachable from some $c_0 \in I$.
- The sets I and D may be infinite.

Symbolic search

- Let $post(C)$ denote the set of immediate successors of a possibly infinite set of configurations C.
- Forward search: Initialize C to I.
- Iterate $C := C \cup post(C)$ until $C \cap D \neq \emptyset$ or a fixed point is reached.
- Question: When is symbolic search effective ?

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.
- $I \in \mathcal{C}$.

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.
- $I \in \mathcal{C}$.
- If $C \in \mathcal{C}$ then $(post(C) \cup C) \in \mathcal{C}$ and is effectively computable.

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.
- $I \in \mathcal{C}$.
- If $C \in \mathcal{C}$ then $(post(C) \cup C) \in \mathcal{C}$ and is effectively computable.
- Emptiness of $C \cap D$ is decidable.

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.
- $I \in \mathcal{C}$.
- If $C \in \mathcal{C}$ then $(post(C) \cup C) \in \mathcal{C}$ and is effectively computable.
- Emptiness of $C \cap D$ is decidable.
- $C_1 = C_2$ is decidable (to check fixpoint is reached).

Sufficient conditions

- Each $C \in \mathcal{C}$ has a finite symbolic representation.
- $I \in \mathcal{C}$.
- If $C \in \mathcal{C}$ then $(post(C) \cup C) \in \mathcal{C}$ and is effectively computable.
- Emptiness of $C \cap D$ is decidable.
- $C_1 = C_2$ is decidable (to check fixpoint is reached).
- Any chain $C_1 \subseteq C_2 \subseteq \ldots$ reaches a fixpoint finitely.

Timed automata

- Variables are clocks: non-negative real valued.
- Transitions guarded by boolean combinations of comparisons with integer bounds, actions reset a subset of clocks.
- Equivalent configurations: when states are the same and values are equivalent with respect to constraints.
- Regions: equivalence classes of configurations.
- Choose $\mathcal{C}$ to be the powerset of regions.

Conditions: regions

- A region can be finitely represented by the set of constraints.

Conditions: regions

- A region can be finitely represented by the set of constraints.
- I is a union of regions.

Conditions: regions

- A region can be finitely represented by the set of constraints.
- I is a union of regions.
- If C is a union of regions, then so is $post(C)$: takes some work.

Conditions: regions

- A region can be finitely represented by the set of constraints.
- I is a union of regions.
- If C is a union of regions, then so is $post(C)$: takes some work.
- Checking emptiness of $C \cap D$: check if C contains some configuration with some state of Q_D as its first element.

Conditions: regions

- A region can be finitely represented by the set of constraints.
- I is a union of regions.
- If C is a union of regions, then so is $post(C)$: takes some work.
- Checking emptiness of $C \cap D$: check if C contains some configuration with some state of Q_D as its first element.
- Checking equality of regions is decidable.

Conditions: regions

- A region can be finitely represented by the set of constraints.
- I is a union of regions.
- If C is a union of regions, then so is $post(C)$: takes some work.
- Checking emptiness of $C \cap D$: check if C contains some configuration with some state of Q_D as its first element.
- Checking equality of regions is decidable.
- Fixedpoint condition follows from the fact that the set of regions is finite.

Lossy channel systems

- Automata extended with unbounded queues.
- Send transitions: no guard, action: add message to channel.
- Receive transitions: guard: non-emptiness of channel; action removes first message.
- Loss transitions: no guard, self loop, removes an arbitrary message.

Symbolic reachability

- Order configurations by the *subword ordering*.
- Choose $\mathcal{C}$ to be all upward closed sets of configurations.
- Forward search does not work, satisfies conditions 1 to 5 but not 6.
- When D is a set of upward closed configurations, backward search works.

Backward symbolic search

- Key idea: Use Higman's lemma to show that any upward closed set can be finitely represented by its set of minimal elements w.r.t. the pointwise order $\geq$.
- Checking that if C is upward closed, so is $pre(C)$ is easy.
- To show that a fixed point is reached in finitely many steps, again appeal to Higman's lemma.

Forward symbolic search

- Choose $\mathcal{C}$ to be the set of simple regular expressions.
- SREs satisfy the first 5 conditions, but the fixpoint cannot be effectively computed.
- One approach: find loops by (a kind of) static analysis (Abdallah et al LICS 99).
- Another: use Angluin's learning algorithms (Varadhan et al FSTTCS 04).

Pushdown Systems

- Natural abstraction of programs written in procedural, sequential languages.

Pushdown Systems

- Natural abstraction of programs written in procedural, sequential languages.
- They generate infinite-state transition systems; states are pairs : (control state, stack content).

Pushdown Systems

- Natural abstraction of programs written in procedural, sequential languages.
- They generate infinite-state transition systems; states are pairs : (control state, stack content).
- Applications: analysis of boolean programs, data-flow analysis, checkpoint algorithms (suspend computations to inspect stack content, for instance, to enforce security requirements).

Automata with stack

- Automata extended with one stack.
- Guards: Check the topmost symbol on stack.
- Actions: replace topmost symbol by a fixed word.
- Configuration (q, v): q holds values of global variables, v holds values of program pointer, values of local variables, return address.

Symbolic reachability

- Choose $\mathcal{C}$ to be the family of regular configurations.
- Each is represented by a DFA.
- I is typically finite and hence regular. Equality of regular sets is decidable.
- If C is regular, showing that $pre(C)$ or $post(C)$ is regular is straightforward.
- Büchi's theorem asserts that the fixedpoint of a chain is regular and can be effectively computed.

Model checking-1

- In fact we often need to verify not only reachability but arbitrary LTL properties.

Model checking-1

- In fact we often need to verify not only reachability but arbitrary LTL properties.
- When valuations are arbitrary – that is, the set of pushdown configurations in which an atomic proposition is true, is an arbitrary subset of the possible ones, model checking is undecidable.

Model checking-2

- When valuations are simple – that is, the truth of an atomic proposition in a pushdown configuration depends only on the control state and topmost stack symbol, we can use a Büchi-like technique to get decidability.

Model checking-2

- When valuations are simple – that is, the truth of an atomic proposition in a pushdown configuration depends only on the control state and topmost stack symbol, we can use a Büchi-like technique to get decidability.
- These techniques can be extended to regular valuations.

Model checking-2

- When valuations are simple – that is, the truth of an atomic proposition in a pushdown configuration depends only on the control state and topmost stack symbol, we can use a Büchi-like technique to get decidability.
- These techniques can be extended to regular valuations.
- Lower bounds: the model checking problem is generically EXPTIME-complete.

Model checking-2

- When valuations are simple – that is, the truth of an atomic proposition in a pushdown configuration depends only on the control state and topmost stack symbol, we can use a Büchi-like technique to get decidability.
- These techniques can be extended to regular valuations.
- Lower bounds: the model checking problem is generically EXPTIME-complete.
- Over pushdown systems, model checking CTL^* reduces to model checking LTL over regular valuations.

LTL:1

- Fix P, a countable set of atomic propositions. LTL formulae are defined by the following syntax:

$$\alpha ::= p \in P \mid \neg\alpha \mid \alpha \vee \beta \mid \bigcirc\alpha \mid \alpha \mathbf{U} \beta$$

- A model is a word $w : \mathcal{N} \rightarrow 2^P$, and the notion $w \models \alpha$ is defined as usual.
- Derived modalities: $\Diamond\alpha = True\mathbf{U}\alpha$ and $\Box\alpha = \neg\Diamond\neg\alpha$.

LTL:2

- Let $\mathcal{L}(\alpha) = \{w | w \models \alpha\}$.
- We know that for every formula α, we can construct a nondeterministic Büchi automaton B_α such that $\mathcal{L}(\alpha) = L(B_\alpha)$, where B_α of size $O(2^{|\alpha|})$.
- Typically, we define a transition system $T = (S, \rightarrow, s_0, V)$ where $V : S \rightarrow 2^P$ is a valuation, and interpret formulas on runs of T. We thus define the model checking problem: $T \models \alpha$, if every infinite run of T satisfies α.

Pushdown systems-1

- A pushdown system is a tuple $S = (C, \Gamma, \Delta, c_0, b)$: C is a finite set of control locations, Γ is the stack alphabet, Δ is the transition relation, c_0 is the initial location and b is the bottom stack symbol.
- $\Gamma \subseteq (C \times \Gamma) \times (C \times \Gamma^*)$, and a transition is written as: $(c, a) \to (d, w)$.
- A configuration is an element of $C \times \Gamma^*$.

Pushdown systems-2

- With a pushdown system S, we associate a transition system T_S with configurations as states, (c_0, b) as the initial state and the transition relation $\Rightarrow$ is the least one satisfying:
  if $(c, a) \longrightarrow (d, w)$ then for all $u \in \Gamma^*$,
  $(c, au) \Rightarrow (d, wu)$.
- Without loss of generality, we assume that b is never removed from stack, and that every transition increases the stack by at most one.

LTL on pushdown systems

Let $S = (C, \Gamma, \Delta, c_0, b)$ be a pushdown system, α an LTL formula, and $V : P \rightarrow 2^{C \times \Gamma^*}$.
The model checking problem comes in three forms:

- Does $(c_0, b) \models \alpha$?
- Is there any configuration that violates α ?
- Is there any reachable configuration that violates α ?

All these problems are undecidable, in general.

Simple valuations

- A set of configurations C is said to be simple if $C \subseteq \{(c, aw) \mid w \in \Gamma^*\}$ for some $c \in C$, $a \in \Gamma$.
- A valuation V is simple, if for every $p \in P$, $V(p)$ is a union of simple sets.

Regular valuations

- A valuation V is said to be regular if for every $p \in P$, $V(p)$ is recognizable and does not contain any configuration with an empty stack.
- Then, for every $p \in P$ and $c \in C$, we have a DFA A_p^c over the alphabet Γ such that $V(p) = \{(c, w) \mid c \in C, w^R \in L(A_p^c)\}$.
- That is, p is true at (c, w) iff A_p^c enters a final state after reading the stack bottom up.

S-automata

- For a PDS $S = (C, \Gamma, \Delta, c_0, b)$, an S-automaton is a tuple $A = (Q, \Gamma, \delta, C, F)$ where Q is a finite set of states, Γ (the stack alphabet of S) is its input alphabet, $\delta : (Q \times \Gamma) \to 2^Q$ is its transition function, C is its set of initial states and F is the set of accepting states.
- δ is extended as usual, and we say that a configiration (c, w) is accepted by A iff $\delta(c, w) \cap F \neq \emptyset$.
- A set of S-configurations C' is regular if it is accepted by some S-automaton.

The main idea

Consider the model checking problem for the initial configuration.

- The problem is reduced to that of emptiness for Büchi pushdown systems.

The main idea

Consider the model checking problem for the initial configuration.

- The problem is reduced to that of emptiness for Büchi pushdown systems.
- The emptiness problem for Büchi pushdown systems is reduced to that of computing the set of predecessors of certain regular sets of configurations.

The main idea

Consider the model checking problem for the initial configuration.

- The problem is reduced to that of emptiness for Büchi pushdown systems.
- The emptiness problem for Büchi pushdown systems is reduced to that of computing the set of predecessors of certain regular sets of configurations.
- The set of predecessors is regular, and an algorithm is given for computing it; this is Büchi's saturation procedure.

Step 1

- Given a PDS $S = (C, \Gamma, \Delta, c_0, b)$, and a formula α, first construct $A_\alpha = (Q, \delta, q_0, F)$ on 2^P.
- Construct the product $B = ((C \times Q), \Gamma, \Delta', (c_0, q_0), b, G)$ by "synchronizing" S and A_α.
- $((c, q), a) \rightarrow' ((c', q'), w))$ if $(c, a) \rightarrow (c', w)$ in S and $q' \in \delta(q, \sigma)$, where σ is the set of propositions true in (c, a).
- Note that we are using the simplicity of valuations here.

Step 2

- Consider a transition $(c, a) \to (c', w)$ in B.
- It is repeating if there exists $v \in \Gamma^*$ such that (c, av) can be reached from (c, a) visiting G.
- Let Rep denoting repeating heads of transitions and let R denote the set $\{(c, aw) \mid (c, a) \in Rep, w \in \Gamma^*\}$.
- We can show that $L(B)$ is nonempty iff $(c_0, b) \in pre^*(R)$.
- Rep is easily computed by an edge marking algorithm.

Regular valuations

- Suppose we have $P_\alpha = \{p_1, \cdots, p_k\}$. Consider all the DFAs M_i^c for each $c \in C$.
- We form a vector of these automata in a canonical fashion with its (product) state from a set $States$.
- The crucial idea is to carry the state vector as part of the stack in a larger pushdown system with $\Gamma' = (\Gamma \times States$.
- Care is needed to ensure consistent configurations.