Well Structured Transition Systems

15th April 2019

Outline

Outline

Infinite States Spaces

- Hardware systems have a fundamental restriction that the amount of hardware described is finite. This leads to possibly very large, but finite state spaces.
- This finite state framework breaks down for software systems with an unbounded domain of variable values. Even a single variable leads to infinitely large number of states.

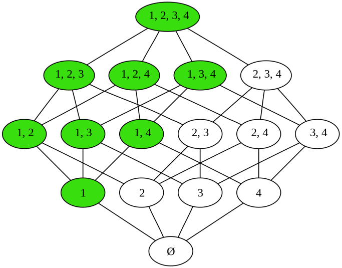
Essentially Finite State Spaces

- The problem of infinite configurations can be addressed by using an abstraction.
- More precisely we define an equivalence relation $\equiv$ on the configurations such that:
  - There are finitely many equivalence classes
  - $\equiv$ is a congruence. That is if, $c_1 \equiv c_2 \vee c_1 \rightarrow c_3$ then there exists c_4, such that, $c_2 \equiv c_4 \vee c_2 \rightarrow c_4$.
- This equivalence relations reduces the infinite state space to a finite one (with the states the number of equivalence classes). We have a bisimulation between the original domain and abstract domain.

Outline

Relaxation of $\equiv$

- Consider a generalization of the $\equiv$ relation as a partial order relation $\preceq$ on the configurations.
- Extending the definition of congruence to this relation we get the notion of *upward-closedness*.



Well Quasi Orders

- Interesting properties arise from the relation $\preceq$ being a Well Quasi Ordering.

WQO

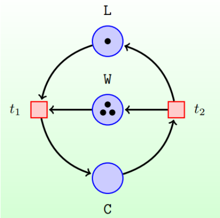
$\preceq$ is a Well Quasi Ordering if for any infinite sequence of elements $c_0, c_1, c_2, \ldots$ there exist indices $i < j$, such that $c_j \preceq c_i$.

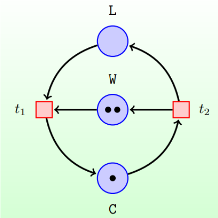
- A consequence of an ordering being a WQO is that all upward closed sets can be expressed as a union of finitely many (principal) filters (Higman).
- Indeed, if this was not the case, then the generators of these filters would produce a contradiction to WQO.

‘

Monotonicity - an example - Petri Nets

- Consider the following transition system that models a mutual exclusion protocol.





- A typical safety property of this system would be - *there is at most one token in the* `C` *state*. The set of states that the above statement describes is upward closed. Hence the problem effectively reduces to the reachability of an upward closed set.

Monotonicity - an example - Petri Nets

Monotonicity and Upward-closedness

- We require the that the transition relation $\rightarrow$ is *monotonic*, that is $c_1 \preceq c_2$ and $c_1 \rightarrow c_3$ implies that $c_2 \rightarrow c_4$ for some $c_4 \rightarrow c_3$.
- Monotonicity implies that upward-closedness is preserved under *Pre*.
    - This gives a scheme for determining the reachability of an upward closed set U.
        1. Initialize $U_0 = U_final$.
        2. Set $U_{i+1} = U_i \cup Pre(U_i)$ till sequence stabilizes.
        3. Return $C_{init} \cap U_{stable} = \phi$?
    - Claim: All U_i are upward closed and above procedure terminates.

Outline

WSTS

- The ideas of monotonicity and WQO combined give a Well Structured Transition System (WSTS or Well Quasi Ordered Transition System) represented as a tuple $(C, \rightarrow, \preceq, C_{init})$.
- Note that each upward closed can be characterized by a finite set of generators. For an upward closed set U, let $gen(U)$ denote this set. If the relation is anti-symmetric, $gen(U)$ is unique.
- Using this characterization we restate the scheme for backward reachability as an algorithm.

WSTS

- Let $c_1 \rightsquigarrow c_2$ stand for $c_2 \in gen(Pre(\uparrow c_1))$.
- For a configuration c, define $(c \rightsquigarrow)$ as $\{c' | c \rightsquigarrow c'\}$ and extend this definition to set of configurations.
- Some observations:(c $\rightsquigarrow$)
  1. $\rightsquigarrow$ is an analog of $Pre()$. More precisely, if $C = gen(U)$, then, $(C \rightsquigarrow) = gen(Pre(U))$.
  2. $U_i \cup Pre(U_i)$ update on up-sets maps to the update $gen(C_i \cup (C_I \rightsquigarrow))$ on their generators.

WSTS

- If $(c \rightsquigarrow)$ is computable and $\preceq$ is decidable then, we can effectively replace the sets U_i, with their generators in the scheme defined earlier.

Algorithm 2 Backward Reachability

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: transition system.
• C_{fin}: finite set of configurations.

Output: Is $\widehat{C_{fin}}$ reachable?

$i \leftarrow 0$
$C_0 := C_{fin}$
repeat
    $C_{i+1} \leftarrow gen\,(C_i \cup (C_i \rightsquigarrow))$
    $i \leftarrow i + 1$
until $C_i \preceq_{\forall\exists} C_{i-1}$
if $\exists c_1 \in C_i \cdot \exists c_2 \in C_{init} \cdot c_1 \preceq c_2$ **then**
    return true
else
    return false
end if

WSTS

- Further optimization's to this algorithm are possible by making observations about $\rightsquigarrow$ which Sriram will discuss in his presentation.

Outline

Unsafe configurations to unsafe traces

- Another way to specify safety properties of the system can be to characterize bad traces (sequences of transitions). For this purpose transitions are labelled with a finite alphabet.
- The resultant system is a composition of finite automata $A = (Q, \delta, q_{init}, F)$ (recognizing traces) and the original transition system $T = (C, \rightarrow, \preceq, C_{init})$. A state is represented by a pair (c, q), $c \in C$ and $q \in Q$.
- The composition is also a WSTS $(C', \rightarrow', \preceq', C'_{init})$ defined as:

  1. $(c_1, q_1) \preceq' (c_2, q_2)$ iff $c_1 \preceq c_2$ and $q_1 = q_2$.
  2. $(c, q) \in C'_{init}$ iff $c \in C_{init}$ and $q \in Q_{init}$

Unsafe configurations to unsafe traces

- Now we have a method transforming checking regular safety properties into reachability of upward-closed sets.
- We construct an automaton $\mathcal{A}$ recognizing language which is complement of safe traces. Compose this with the given WSTS. Check if the accepting states for $\mathcal{A}$ are reached. *The set of these states is upward closed*.

Algorithm 4 Checking Safety Properties

Input: • $\mathcal{T} = (C, \longrightarrow, \preceq, C_{init})$: LTS.
• Σ: regular set of words over A.

Output: $\mathit{Traces}(\mathcal{T}) \subseteq \Sigma$?

1: construct $\mathcal{A}$ s.t. $\mathit{Lang}(\mathcal{A}) = \neg\Sigma$
2: $\mathcal{T}' \leftarrow (\mathcal{T} \| \mathcal{A}) = \left(C', \longrightarrow', \preceq', C'_{init}\right)$
3: $C_{fin} \leftarrow \{(c, q) \mid c \in \mathit{gen}(C) \wedge q \in Q_{fin}\}$.
4: use Algorithm 3 to check whether $\widehat{C_{fin}}$ is reachable.

Outline

Applications

- Petri Nets ... As discussed in the example earlier
  1. $\preceq$ is the natural element-wise ordering on the number of tokens in each place in the net, which is a WQO (Dickson's lemma)
  2. $\rightarrow$ is determined by the firing transitions
  3. We also have computability of $(c \rightsquigarrow)$ and decidability of $\preceq$
- Lossy Channel Systems ... Finite state transition automata are augmented by a set of channels on which the automata can read and write tokens. There are no guarantees about the channel as tokens may be lost non-deterministically.
  1. $\preceq$ is the sub-word ordering (for each channel). The fact that this is a WQO follows from Higman's lemma.
  2. $\rightarrow$ is given by $\{\mathit{silent}$ transitions$\} \cup \texttt{read} \cup \texttt{write}$ actions to the channels

Outline

Comparison

- The above *backward-reachability* methods fall under the class of set saturation methods. We perform iterative updates on upward-closed sets (or their generators).
- Another method is to consider *forward-reachability* using structures such as coverability trees. This class of methods is called tree-saturation methods.

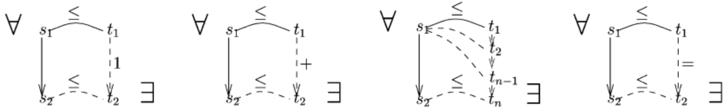
Finite Reachability Tree

Given a WSTS, $(C, \rightarrow, \preceq, C_{init})$ consider a tree defined as follows:

- nodes are represented as $(n : c)$ (labelled with configurations) and flagged as either *dead* or *live*
- a leaf is dead (has no children) while a live node $(n : c)$ has its $Post(c)$ set as its children
- if a node $(n_1 : c_1)$ has a node $(n_2 : c_2)$ as its strict descendant with $c_1 \preceq c_2$ then we say that n_1 subsumes n_2 (in set terms c_2 is in upward closure of c_1 and hence keeping track of n_1 is sufficient for reachability)
- the leaves are exactly the set of subsumed nodes and terminal nodes

Finite Reachability Tree

Further results need slightly restricted notions of compatibility: transitive, stuttering, strict compatibility



(a) strong (b) transitive (c) stuttering (d) reflexive

Fig. 3. Strict compatibility.

Figure: compatibility in WSTS

Finite Reachability Tree

Termination

Proposition 4.5. *$\mathscr{S}$ has a non-terminating computation starting from s iff $FRT(s)$ contains a subsumed node.*

Theorem 4.6. *Termination is decidable for WSTSs with* (1) *transitive compatibility,* (2) *decidable $\leqslant$, and* (3) *effective Succ.*

Boundedness

Proposition 4.10. *For any $s \in S$, $Succ^*(s)$ is infinite iff $FRT(s)$ contains a leaf node $n : t$ subsumed by an ancestor $n' : t$ with $t' < t$.*

Theorem 4.11. *The boundedness problem is decidable for WSTSs with* (1) *strict transitive compatibility,* (2) *a decidable $\leqslant$ which is a partial ordering, and* (3) *computable Succ.*

Further discussions on board.

References I

Abdulla, Parosh Aziz
Well (and better) quasi-ordered transition systems
Bulletin of Symbolic Logic, 2010

A. Finkel, Ph. Schoebelen
Well structured transition systems everywhere!
Theoretical Computer Science 2001

Thank You!