

CIS 673: Lecture 5: Sept 20  
Graph Reachability

- Classical graph search: explore successors of each state in a systematic manner
- Successor region of a state:  $post_G(s)$  is the set of successors of  $s$ .
- Predecessor region of a state:  $pre_G(s)$  is the set of  $t$  such that  $t \rightarrow s$
- Solving reachability problem  $(G, \sigma^I)$  corresponds to checking if  $post^*(\sigma^I) \cap \sigma^T$  is empty, or equivalently, if  $pre^*(\sigma^T) \cap \sigma^I$  is empty
- Forward or backward search
- Enumerative search: explore one state at a time

## Depth-first Search Algorithm

```
input  $G$ : enumgraph;  $\sigma^T$ : enumreg;
local  $\sigma^R$ : enumreg;  $\tau$ : stack of state;  $t$ : state
begin
   $\sigma^R := \text{EmptySet}$ ;  $\tau := \text{EmptyStack}$ ;
  foreach  $t$  in  $\text{InitQueue}(G)$  do
    if DFS( $t$ ) then return  $\text{Reverse}(\tau)$ 
  return Done
end.

function DFS: bool
input  $s$ : state; local  $t$ : state;
begin
   $\tau := \text{Push}(s, \tau)$ ;
  if not  $\text{IsMember}(s, \sigma^R)$  then
    if  $\text{IsMember}(s, \sigma^T)$  then return true
     $\sigma^R := \text{Insert}(s, \sigma^R)$ ;
    foreach  $t$  in  $\text{PostQueue}(s, G)$  do
      if DFS( $t$ ) then return true
   $\tau := \text{Pop}(\tau)$ 
  return false
end.
```

## Complexity of reachability

- Classical graph representation
  - Initial region is a queue of states
  - Transitions: Array indexed by states, for each state  $s$ , a queue for  $post(s)$
  - Target region: boolean array indexed by states
  - Reachable region: boolean array indexed by states
- Time complexity:  $O(n + m)$  for  $n$  states and  $m$  transitions
- Complexity class of graph reachability: NLOGSPACE

## Invariant Verification

- How does invariant verification differ from graph reachability?
- Input is the module description, and not the graph (graph is implicitly represented)
- Naive solution: If  $P$  is finite, then construct the transition graph  $G_P$ , and apply reachability algorithm
- $G_P$  may be too large, classical representation not suitable
- To make complexity analysis precise, let us look at classes of modules

## Propositional Modules

- All variables are propositions (booleans)
- All expressions (in guards and assignments) are propositional formulas

$$p ::= x \mid true \mid false \mid p \vee p \mid p \wedge p \mid \neg p \mid p \rightarrow p \mid p \leftrightarrow p,$$

- Target region is also a propositional formula
- If  $P$  has  $k$  vars then  $G_P$  can have  $2^k$  states and  $4^k$  transitions
- Complexity of depth-first search is exponential for propositional invariant verification problem

## Enumerated Modules

- All variables have enumerated types
- All guards are enumerated formulas

$$p ::= x = m \mid x \neq m \mid \text{true} \mid \text{false} \mid p \vee p \mid p \wedge p,$$

$m$  is constant

- All assignments are to variables or constants
- Invariant is an enumerated formula
- Example: mutual exclusion (Pete)

## State-space Explosion

- Number of reachable states can grow exponentially with the number of module variables
- Central theme of research in model checking, and much of this course: How do we overcome state-space explosion?

## On-the-fly Search

- Do not construct the graph  $G_P$  a priori, but only as needed
- Number of reachable states may be much smaller than the number of all possible states
  - Init takes the module description and returns a queue of initial states
  - Post takes the module description and a state  $s$ , and returns a queue of successors of  $s$
- Target region (negation of invariant) represented as a propositional formula Only implement the membership query  $IsMember(s, \sigma^T)$

## On-the-fly Search

- Advantages:
  - Space optimization: only reachable states are stored
  - Graph need not be finite, only required to be finitely branching
  - May terminate early if target is found to be reachable
- Implementation Issue: how to store the guarded commands of module so that Post can be computed quickly
  - Presort the atoms in a consistent order
  - Usually a variable like *pc* can be used to structure the guarded assignments

## Hashing

- How to store states visited so far? Data structure for  $\sigma^R$ :
  - Required operations: IsMember and Insert
  - Space should be small, ideally proportional to number of reachable states (rather than all possible states)
- Hash-table:
  - Hashing function that maps a state to  $1 \dots N$
  - Array of size  $N$  whose  $i$ -th entry is a list of states (different states may be hashed to the same index)
- Performance depends on choice of hash-function and  $N$

## More Space Saving

- Natural representation of state: record type with a field corresponding to each variable.  
Compression to reduce number of bits
- Bit-state hashing: what if  $N$  is limited by available memory and is small compared to reachable states?
  - Each entry of hash-table is a bit
  - Initially 0, and changed to 1 when some state with hash-index  $i$  is stored
  - Hash-collisions undetected: if two states get hashed to same index, then only the first one is searched
  - Verification approximate: cannot say *Done* for sure
  - Super-trace algorithm (SPIN):  
Use two independent hash-functions and two bits per entry (Good probabilistic bounds)

## PSPACE Complexity

- We have an exponential algorithm for invariant verification of propositional/enumerated modules. Can we prove a lower bound?
- Theorem: The propositional invariant-verification problem is PSPACE-hard.
- Proof: By reduction from the halting problem for space-bounded Turing machines
- PSPACE-hardness applies both to
  - Single atom with many variables
  - Many atoms each of constant size

## Complexity Classes

- Classes defined by space and time requirements of Turing machines
- Hierarchy:  
$$NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$
- Canonical NP problem: satisfiability of propositional formulas
- Canonical PSPACE problems
  - Reachability in an exponential size graph
  - Quantified propositional formulas
- For NP, small witnesses exist, good heuristics are more likely. For PSPACE, witnesses need not be small, but can be constructed on-the-fly using small space
- EXPTIME: even harder, explicit construction of the exponential graph required (eg. solving AND-OR games in an exponential graph)

## PSPACE membership

- Solving reachability problem in a graph with  $n$  vertices using depth-first search requires space  $O(n)$
- Solving propositional invariant verification using depth-first search requires exponential space
- Is there a PSPACE algorithm?
  - Binary search can be used to save space
  - General: A graph with  $f(n)$  vertices can be searched in space  $O(\log (f(n)))$
  - Caution: this is not a practical algorithm, only to establish complexity class

## PSPACE Algorithm

Input: Propositional module  $P$  and state  $s$

Output: Yes if  $s$  is reachable.

foreach state  $t$  do

  if  $t \in \sigma_P^I$  then

    if  $\text{Check}(t, s, 2^n)$  then return Yes

  return No.

function  $\text{Check}(t, u, i)$ : bool

Input: states  $t$  and  $u$  and integer  $1 \leq i \leq 2^n$ .

Output: Is there trajectory from  $t$  to  $u$  of  
  length  $\leq i$ .

  if  $t = u$  then return true

  if  $t \rightarrow_P u$  then return true

  if  $i > 2$  then

    foreach state  $v$  do

      if  $\text{Check}(t, v, \lfloor i/2 \rfloor + 1)$  then

        if  $\text{Check}(v, u, \lfloor i/2 \rfloor)$

          then return true

  return false.

## Compositional Verification

- Can we decompose the verification problem, and check invariants of components?
- Intuitively, compositionality means that from a property of a component, we can infer a property of the system
- Compositionality Theorem:
  - An invariant of  $P$  is an invariant of  $P \parallel Q$
  - An invariant of  $P$  is an invariant of **hide**  $x$  **in**  $P$
  - If  $p$  is an invariant of  $P$ , then  $p[\rho]$  is an invariant of the renamed module  $P[\rho]$ .

## Using Compositionality

- Goal: To prove that  $p$  is an invariant of  $P \parallel Q$
- Assumption: Verifying  $P$  or  $Q$  in isolation is less expensive than verifying  $P \parallel Q$
- Naive strategy: Establish that
  - $p$  is an invariant of  $P$
  - $p$  is an invariant of  $Q$
- Likely to fail:  $P \parallel Q$  is more constrained, and satisfies a lot more invariants than  $P$  or  $Q$  in isolation
- Does  $P_1$  of *Pete* satisfy

$$\text{mutex: } \neg(pc_1 = inC \wedge pc_2 = inC)$$

## Refined Strategy for Compositionality

- Goal: To prove that  $r$  is an invariant of  $P \parallel Q$
- Find two predicates  $p$  and  $q$  such that
  - $p$  is an invariant of  $P$
  - $q$  is an invariant of  $Q$
  - $p \wedge q \rightarrow r$  is valid
- Interactive verification: Designer uses his/her knowledge/intuition to simplify the problem
- A more useful strategy based on assume-guarantee reasoning is supported by Mocha, but will be discussed later

## Application: Railroad Controller

- Goal: Establish that

$$safe: \quad \neg(pc_W = bridge \wedge pc_E = bridge)$$

is an invariant of the compound module

$$\begin{aligned} RailroadSystem = & \mathbf{hide} \text{ } arrive_W, arrive_E, leave_W, leave_E \mathbf{in} \\ & \parallel Train_W \\ & \parallel Train_E \\ & \parallel Controller2. \end{aligned}$$

- Can we look at one train at a time?
- How did we design the controller?
- When  $Train_W$  is on the bridge, we expect  $signal_W$  to be green and  $signal_E$  to be red

## Applying Compositionality

- Establish that

$$safe_W: \quad pc_W = bridge \rightarrow s_W = green \wedge s_E = red$$

is an invariant of  $Train_W \parallel Controller2$

- Conclude, by renaming,

$$safe_E: \quad pc_E = bridge \rightarrow s_E = green \wedge s_W = red$$

is an invariant of  $Train_E \parallel Controller2$

- By compositionality,  $safe_W \wedge safe_E$  is an invariant of  $Train_W \parallel Train_E \parallel Controller2$
- $safe_W \wedge safe_E \rightarrow safe$  is valid
- $safe$  is an invariant of  $Train_W \parallel Train_E \parallel Controller2$
- By compositionality of hiding,  $safe$  is an invariant of RailroadSystem.

## Symbolic Reachability

- Instead of enumerating states, compute with regions (state-sets) that are represented symbolically (eg.  $20 \leq x \leq 99$ )
- Implicit representation of transition relation and reachable states
- For propositional or enumerated modules, binary decision diagrams serve as a compact representation
  - Binary decision diagrams: representation for boolean functions due to Bryant [87]
  - Application to model checking: tool SMV by McMillan [91]
  - Popular in hardware applications
- Symbolic model checking recently won ACM's Theory in Practice award

## Symbolic Data Types

- Symbolic regions support following operations
  - $\cup: \mathbf{symreg} \times \mathbf{symreg} \mapsto \mathbf{symreg}$
  - $\cap: \mathbf{symreg} \times \mathbf{symreg} \mapsto \mathbf{symreg}$
  - $=: \mathbf{symreg} \times \mathbf{symreg} \mapsto \mathbf{bool}$
  - $\subseteq: \mathbf{symreg} \times \mathbf{symreg} \mapsto \mathbf{bool}$
  - *EmptySet*:  $\mathbf{symreg}$
- Note: No enumeration, so number of states in a region is not an issue
- Symbolic transition graph supports
  - *InitReg*:  $\mathbf{symgraph} \mapsto \mathbf{symreg}$
  - *PostReg*:  $\mathbf{symreg} \times \mathbf{symgraph} \mapsto \mathbf{symreg}$

## Symbolic Search

Input: a transition graph  $G$ , and a region  $\sigma^T$

Output: answer to reachability problem  $(G, \sigma^T)$ .

input  $G$ : symgraph;  $\sigma^T$ : symreg;

local  $\sigma^R$ : symreg;

begin

$\sigma^R := \text{InitReg}(G)$ ;

repeat

if  $\sigma^R \cap \sigma^T \neq \text{EmptySet}$  then

return Yes;

if  $\text{PostReg}(\sigma^R, G) \subseteq \sigma^R$  then

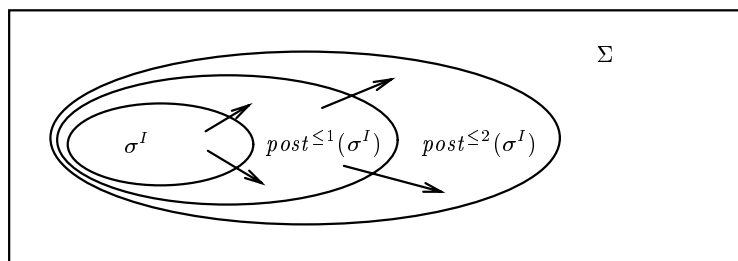
return No;

$\sigma^R := \sigma^R \cup \text{PostReg}(\sigma^R, G)$

forever

end.

## Symbolic Search



Like breadth-first search, guaranteed to terminate if the diameter of the graph is bounded: there is  $i$  such that every reachable state is reachable within  $i$  transitions from some initial state