Embedded Systems Specification and Design

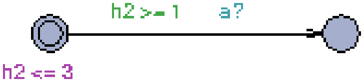
David Kendall

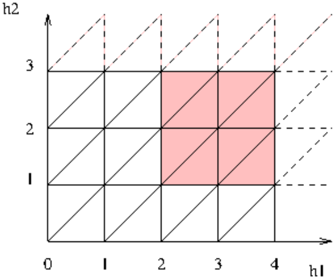
Northumbria University

State space of timed automaton is infinite ?!!

- Introduction of real-valued clock variables to FSM makes its state space infinite
- How can we exhaustively explore infinite state space?
  - We can't
- How can we exhaustively explore state space of timed automaton?
  - Find an equivalent finite representation of its state space
- This is the key idea that makes model-checking of timed automata possible
- Finite representations
  - Region graph
  - Zone (simulation) graph

Zones and their representation



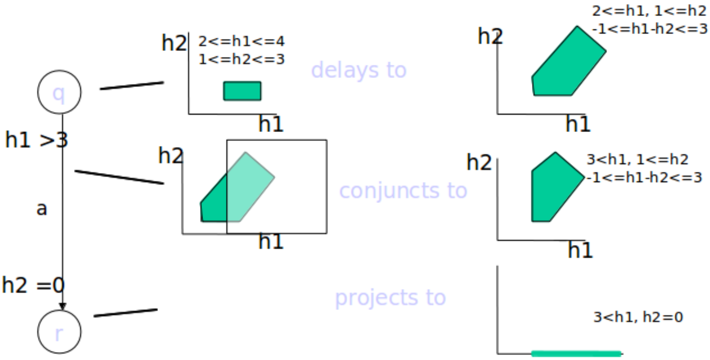


M	h_0	h_1	h_2
h_0	$(0, \leq)$	$(-2, \leq)$	$(-1, \leq)$
h_1	$(4, \leq)$	$(0, \leq)$	$(\infty, <)$
h_2	$(3, \leq)$	$(\infty, <)$	$(0, \leq)$

M'	h_0	h_1	h_2
h_0	$(0, \leq)$	$(-2, \leq)$	$(-1, \leq)$
h_1	$(4, \leq)$	$(0, \leq)$	$(3, \leq)$
h_2	$(3, \leq)$	$(1, \leq)$	$(0, \leq)$

DIFFERENCE BOUND MATRICES

Operations on zones



Thus (q,2<=h1<=4,1<=h2<=3) =a => (r, 3<h1, h2=0)

Symbolic states and transitions

Symbolic state

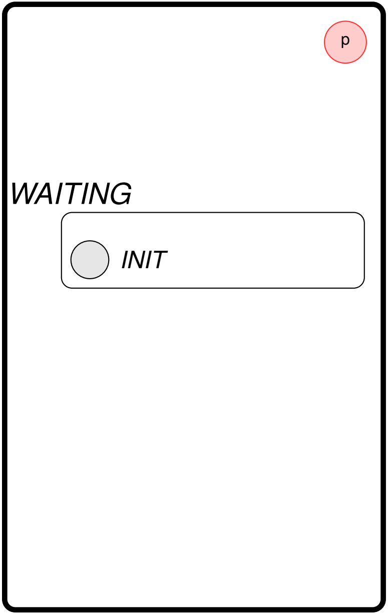
A symbolic state, $s = (q, \zeta)$, consists of:

- *a location vector*, q, and
- *a zone*, ζ

Symbolic transition

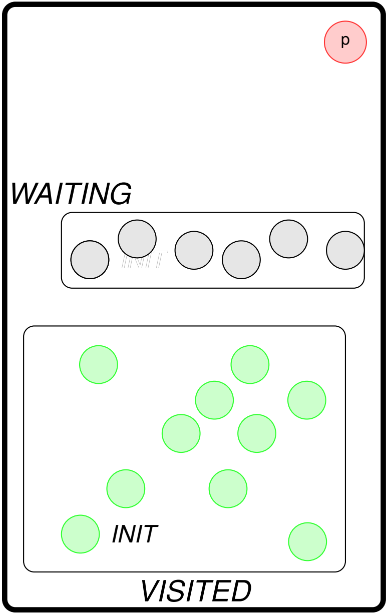
If the edges of a network of timed automata justify a transition from location vector q to location vector q' and the delays to, conjuncts and projects operations transform the zone ζ to ζ', then for $s = (q, \zeta)$ and $s' = (q', \zeta')$, we say there is a *symbolic transition* from s to s' and write $s \Rightarrow s'$

Forward reachability



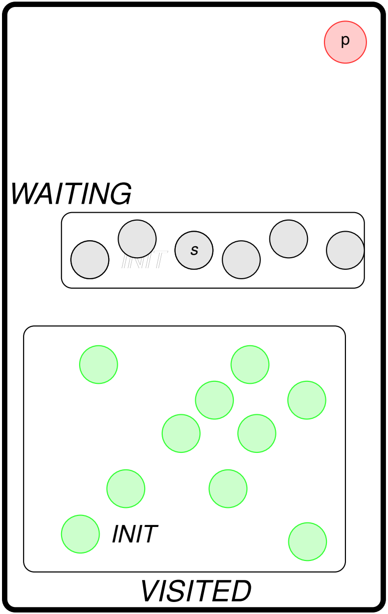
$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $s \notin VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

Forward reachability



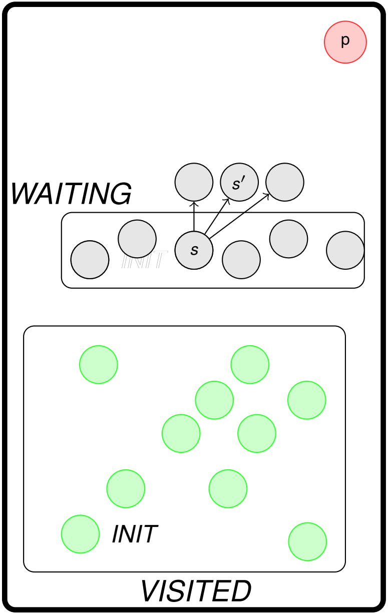
$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $s \notin VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

Forward reachability



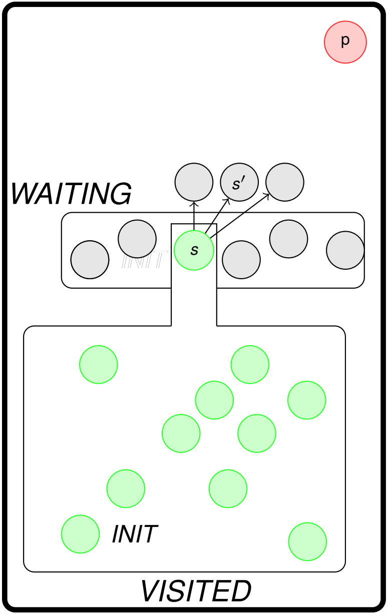
$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
remove some $s = (q, \zeta)$ from $WAITING$
if s satisfies p **then**
return true
else
if $s \notin VISITED$ **then**
$VISITED \leftarrow VISITED \cup \{s\}$
$SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
$WAITING \leftarrow WAITING \cup SUCC$
end if
end if
end while
return false

Forward reachability



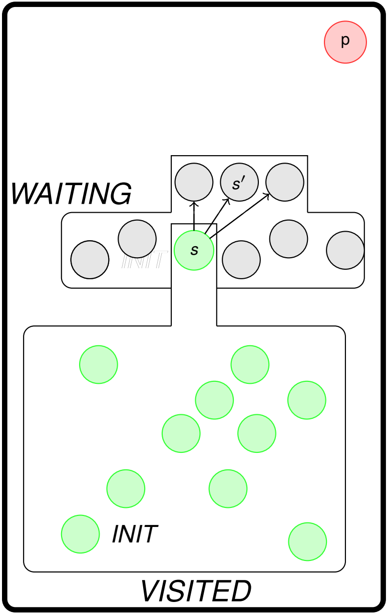
$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $s \notin VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

Forward reachability



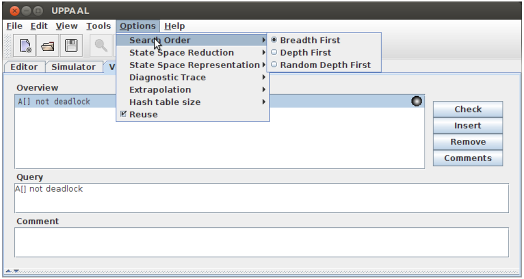
$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $s \notin VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

Forward reachability



$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $s \notin VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

Search order



Breadth First:

- Search state space in breadth-first order
- Good for complete search and shortest/fastest trace

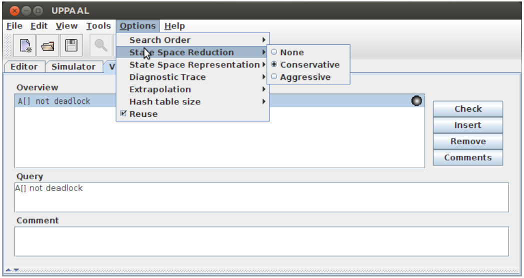
Depth First:

- Search state space in depth-first order
- Better than breadth-first if counter-example expected

Random Depth First:

- Like depth-first order but randomised choice of next edge

State space reduction

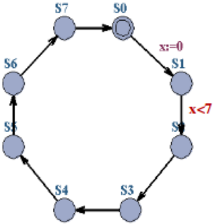


- Inclusion abstraction
- Active-clock reduction
- …

Inclusion abstraction

$INIT \leftarrow (q^0, \text{zero})$
$WAITING \leftarrow \{INIT\}$
$VISITED \leftarrow \emptyset$
while $WAITING \neq \emptyset$ **do**
  remove some $s = (q, \zeta)$ from $WAITING$
  if s satisfies p **then**
    return true
  else
    if $\nexists \zeta' \supseteq \zeta \bullet (q, \zeta') \in VISITED$ **then**
      $VISITED \leftarrow VISITED \cup \{s\}$
      $SUCC \leftarrow \{s' \mid s \Rightarrow s'\}$
      $WAITING \leftarrow WAITING \cup SUCC$
    end if
  end if
end while
return false

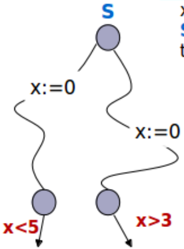
Active clock reduction



x is only ***active*** in location **S1**

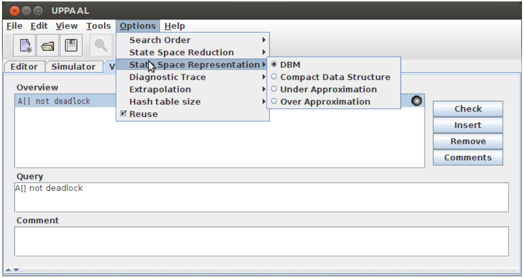
Definition

x is ***inactive*** at **S** if on all path from **S**, x is always reset before being tested.



- Inactive clocks need not be stored
- ⇒ smaller DBMs
- ⇒ smaller state space

State space representation



Under-approximation

- Uses bit-state hashing
- Does not guarantee to explore all states
- So, if verifier claims that some state is unreachable, it may be wrong
- Useful mainly for discovering bugs, not for proving their absence

Over-approximation

- Uses convex hull abstraction
- May explore too many states
- So, if verifier claims that some state is reachable, it may be wrong
- Can prove absence of bugs but when it fails you need to ask if it's a genuine failure or a failure caused by the abstraction

State space explosion summary

- Infinite number of states can be represented finitely by using symbolic representation, e.g. *zones*
- State space explosion
  - State space may be finite but often still too large to handle with current computing power
- Attacks on state space explosion
  - Store fewer states
  - Store smaller states

And finally. . .

Much recent research has pushed model-based development using timed automata into new areas

- Priced timed automata
  - Reason about resource usage, e.g. energy
- Timed games
  - Controller synthesis
- Probabilistic timed automata
  - Reason about probabilistic systems, e.g. Bluetooth
- Statistical model-checking
  - Don’t explore full state space, but use statistical techniques to calculate confidence in results based on partial exploration