CS477 Formal Software Dev Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures
by Mahesh Vishwanathan, and by Gul Agha

April 19, 2020

LTL Büchi Automaton

- Problem: How to convert an LTL formula in a Büchi Automaton
- Assume LTL formula φ in reduded form
- Need
  - finite *alphabet* Σ
  - finite set of *states* S
  - *transition relation* Δ
  - *start states* I
  - *labeling* of the *states* L
  - *accepting states* F

Nodes for building Büchi Automaton

- States will be natural numbers
- As we build the graph, need to keep temp information
- First pass: Label each node with:
  - *Name*: Unique number for the node.
  - *Incoming*: Set of sates with edges that point to current node.
  - *New*: Set of subformulae of φ that must hold at the current node and have not been processed yet.
  - *Old*: Set of subformulae of φ that must hold at the current node and have been processed.
  - *Next*: A set of subformulae of φ that must hold at every immediate successors of the current state.

Input to Algorithm

- Main function expand
- Defined iteratively
- Takes current node, set of nodes previously created, next state number
- Main idea: Separate φ it what holds in current state, and what holds in next state using

$$\varphi \mathcal{U} \psi = \psi \vee (\varphi \wedge \circ(\varphi \mathcal{U} \psi))$$

and

$$\varphi \mathcal{V} \psi = \psi \wedge (\varphi \vee \circ(\varphi \mathcal{V} \psi))$$

- Will define expand imperatively
- Need to convert to functional to define in Isabelle

Helper Functions: SF, New1, New2,Next1

- SF calculates all subformulae of an LTL formula

Formula	New1	Next1	New2
$\varphi \mathcal{U} \psi$	$\{\varphi\}$	$\{\varphi \mathcal{U} \psi\}$	$\{\psi\}$
$\varphi \mathcal{V} \psi$	$\{\psi\}$	$\{\varphi \mathcal{V} \psi\}$	$\{\varphi, \psi\}$
$\varphi \wedge \psi$	$\{\varphi, \psi\}$	$\emptyset$	$\emptyset$
$\varphi \vee \psi$	$\{\varphi\}$	$\emptyset$	$\{\psi\}$
$\bigcirc \varphi$	$\emptyset$	$\{\varphi\}$	$\emptyset$

expand: End case merge

- If *New* of current node is emtpty, then we want to combine current node with nodes previously created. Two cases, handled by merge.
- Input to merge:
  - current node,
  - existing node not yet tried,
  - existing nodes that failed to merge with current node,
  - next number to use to make the next state
- First case: No nodes previously created left with which to try to merge :

$$
\begin{array}{l}
\text{merge}\ (\mathit{node},\ \mathit{Nodes\_Set},\ \mathit{next\_node\_num},\ \mathit{node\_set\_seen}) = \\
\quad \text{case}\ \mathit{Nodes\_Set}\ \text{of} \\
\quad \mathit{Nodes\_Set} = \{\,\} \Rightarrow \\
\quad \text{expand}\ (\mathit{next\_node\_num},\ \{\text{Name}(\mathit{node})\},\ \text{Next}(\mathit{node}),\ \{\,\},\ \{\,\}) \\
\qquad\qquad ((\{(\text{Name}(\mathit{node}), \text{Incoming}(\mathit{node}), \text{Old}(\mathit{node}), \text{Next}(\mathit{node}))\} \cup \\
\qquad\qquad\quad \mathit{node\_set\_seen}) \\
\qquad\qquad\ (\mathit{next\_node\_num} + 1))
\end{array}
$$

expand: End case merge, second case

- Second case: Some previously existing nodes haven't been tried

$$
\begin{array}{l}
\mathit{Nodes\_Set} = (\{\,(\mathit{name}, \mathit{incoming}, \mathit{old}, \mathit{next})\,\} \uplus \mathit{more\_nodes}) \Rightarrow \\
\quad \text{if } (\mathsf{Old}(\mathit{node}) = \mathit{old}) \wedge (\mathsf{Next}(\mathit{node}) = \mathit{next}) \\
\quad \text{then} \\
\quad\quad (\mathit{node\_set\_seen} \cup \{(\mathit{name},\ (\mathsf{Incoming}(\mathit{node}) \cup \mathit{incoming}),\ \mathit{old},\ \mathit{next})\} \cup \\
\quad\quad\quad \mathit{more\_nodes}), \\
\quad\quad\ \mathit{next\_node\_num}) \\
\quad \text{else} \\
\quad\quad \text{merge}\ (\mathit{node}, \\
\quad\quad\quad\quad\quad\quad \mathit{more\_nodes}, \\
\quad\quad\quad\quad\quad\quad \mathit{next\_node\_num}, \\
\quad\quad\quad\quad\quad\quad (\{(\mathit{name},\ \mathit{incoming},\ \mathit{old},\ \mathit{next})\} \cup \mathit{node\_set\_seen}))
\end{array}
$$

expand case: New(*node*) is empty

function expand (node, (*Nodes_Set*, next_node_num)) =
  case New(*node*) *of*
    New(*node*) = { } ⇒
      merge (*node*, *Nodes_Set*, *next_node_num*, { })
    New(*node*) = {η} ⊎ *more_new* ⇒
      New(*node*) := *more_new*;
      let *more_old* := Old(*node*) ∪ {η} *in*
      Old(*node*) := *more_old*;
      case η of

expand case: atomic propostions and their negations

case η of
$\eta =$ A, or $\neg$A, where A proposition, or $\eta =$ true, or $\eta =$ false $\Rightarrow$
if $\eta =$ false or $\neg\eta \in$ more_old
then return(*Nodes_Set*, *next_node_num*)
else return (expand ((Name(*node*), Incoming(*node*),
more_new, *more_old*, Next(*node*)),
(*Nodes_Set*, *next_node_num*))

expand case: η equiv to or

$\eta = \varphi\,\mathcal{U}\,\psi$, or $\varphi\,\mathcal{V}\,\psi$, or $\varphi \vee \psi \;\Rightarrow$

let s_1 := (Name(*node*), Incoming(*node*),
more_new $\cup$ ({New1(η)} $\setminus$ *more_old*),
more_old, Next(*node*) $\cup$ {Next1(η)}) in

let s_2 := (*next_node_num*, Incoming(*node*),
more_new $\cup$ ({New2(η)} $\setminus$ *more_old*),
more_old, Next(*node*)) in

return(expand (s_2, (expand (s_1, (*Nodes_Set*, (*next_node_num* + 1)

expand cases: and and next

$\eta = \varphi \wedge \psi \ \Rightarrow$

return(expand ((Name(*node*), Incoming(*node*),
more_new $\cup$ ($\{\varphi, \psi\} \setminus$ *more_old*),
more_old, Next(*node*)),
(*Nodes_Set*, *next_node_num*)))

$\eta = \circ\varphi \ \Rightarrow$

return(expand ((Name(*node*), Incoming(*node*),
more_new, *more_old*, Next(*node*) $\cup \{\varphi\}$),
(*Nodes_Set*, *next_node_num*)))

function create_graph(μ) =
return(expand ((1, $\{0\}$, $\{\mu\}$, $\{\,\}$, $\{\,\}$), ($\{\,\}$, 2)))