CS 267: Automated Verification

Lecture 9: LTL Buchi Automata Translation

Instructor: Tevfik Bultan

LTL

- We are going to discuss LTL to Buchi automata translation
- First let's recall LTL semantics
- I will also add a new operator called R (release) to make the translation to Buchi automata easier

LTL Semantics

Given an execution path x and LTL properties p and q

$x \models p$	iff	$L(x_0, p) = \text{True}$, where $p \in AP$
$x \models \neg p$	iff	not $x \models p$
$x \models p \wedge q$	iff	$x \models p$ and $x \models q$
$x \models p \vee q$	iff	$x \models p$ or $x \models q$

$x \models X\, p$	iff	$x^1 \models p$
$x \models G\, p$	iff	for all $i \geq 0$, $x^i \models p$
$x \models F\, p$	iff	there exists an $i \geq 0$ such that $x^i \models p$
$x \models p\, U\, q$	iff	there exists an $i \geq 0$ such that $x^i \models q$ and for all $0 \leq j < i$, $x^j \models p$
$x \models p\, R\, q$	iff	for all $j \geq 0$, if for all $0 \leq i < j$, $x^i \not\models p$ then $x^j \models q$

LTL Equivalences

- Given an LTL formula convert it to positive normal form:
  - Negations are only applied to atomic propositions (there is no negation outside of a temporal operator)
- Use the following equivalences to translate the LTL formulas to positive normal form:

$\neg(p \; U \; q) \equiv \neg p \; R \; \neg q$

$\neg(p \; R \; q) \equiv \neg p \; U \; \neg q$

$\neg(X \; p) \equiv X \; \neg p$

$\neg(p \; R \; q) \equiv \neg p \; U \; \neg q$

$F \; p \equiv true \; U \; p$

$G \; p \equiv false \; R \; p$

LTL Buchi Automata Translation

[Gerth, Peled, Vardi, Wolper 95]

- *Each state of the automata will store a set of properties that should be satisfied on paths starting at that state*
  - These properties will be stored in lists called *Old* and *New* where Old means already processed and New means still needs to be processed

- *Each state will also store a set of properties which should be satisfied on paths starting at the next states of that state*
  - These properties will be stored in the list *Next*

- The incoming transitions for a state will be stored in the list Incoming

LTL Buchi Automata Translation

- We will start with a node which has the input LTL property in its New list

- We will process the formulas in the New list of each node one by one

  - When we have f U g in the New list we will use
    $f \ U \ g \equiv g \vee (f \wedge X\,(f \ U \ g))$

  - When we have f R g in the New list we will use
    $f \ R \ g \equiv g \wedge (f \vee X\,(f \ R \ g))$

LTL Buchi Automata Translation

- When we process a formula from a node we will either replace the node with a new node or we will replace it with two new nodes (i.e., we will split it to two nodes)

  – When a node q is replaced by a node q' we will have

$$(\text{Old}(q) \wedge \text{New}(q) \wedge X\ \text{Next}(q)) \Leftrightarrow (\text{Old}(q') \wedge \text{New}(q') \wedge X\ \text{Next}(q'))$$

Means conjunction of all the formulas in these lists

  – When a node q is split into two nodes q_1 and q_2 we will have

$$(\text{Old}(q) \wedge \text{New}(q) \wedge X\ \text{Next}(q))$$
$$\Leftrightarrow((\text{Old}(q_1) \wedge \text{New}(q_1) \wedge X\ \text{Next}(q_1))$$
$$\vee (\text{Old}(q_2) \wedge \text{New}(q_2) \wedge X\ \text{Next}(q_2)))$$

```
Translate(f) { expand([Incoming:={init}, Old:=∅, New:={f}, Next:=∅], ∅) }

Expand(q, NodeList) {
If New(q) is empty
then
    if there exists a node r in NodeList s.t. Old(r) = Old(q) and Next(r) = Next(q)
    then  Incoming(r) := Incoming(q) ∪ Incoming(r);
          return(NodeList);
    else  create a new node q' s.t.   Incoming(q')=q, Old(q')= ∅,
                                      New(q')=Next(q), Next:=∅;
          return expand(q', Nodelist ∪ {q});
else // New(q) is not empty
    pick a formula f from New(q) and remove it from New(q);
    if f is already in Old(q) then return expand(q, Nodelist);
    else if (f ∈ AP or Neg(f) ∈ AP or f is a boolean constant)
          then if (f≡false or Neg(f) ∈Old(q)) then return(Nodelist);
          else  create a node q' s.t.
                    Incoming(q')=Incoming(q),
                    Old(q')=Old(q) ∪ {f},
                    New(q')=New(q) – {f},
                    Next(q')=Next(q);
                return expand(q', Nodelist);
```

```
else if (f ≡ h ∨ k)
        create two nodes q1 and q2 s.t
                Incoming(q1) = Incoming(q2) = Incoming(q),
                Old(q1) = Old(q2) = Old(q) ∪ {h ∨ k},
                New(q1) = (New(q) – {h ∨ k}) ∪ {h},
                New(q2) = (New(q) – {h ∨ k}) ∪ {k},
                Next(q1) = Next(q2) = Next(q);
        return expand(q2, expand(q1, Nodelist));
else if (f ≡ h ∧ k)
        create a node q' s.t.
                Incoming(q')=Incoming(q),
                Old(q')=Old(q) ∪ {h ∧ k},
                New(q')=(New(q) – {h ∧ k}) ∪ {h} ∪ {k},
                Next(q')=Next(q);
        return expand(q', Nodelist);
else if (f ≡ X h)
        create a node q' s.t.
                Incoming(q')=Incoming(q),
                Old(q')=Old(q) ∪ {X h},
                New(q')=New(q) ) – {X h},
                Next(q')=Next(q) ∪ {h};
        return expand(q', Nodelist);
```

else if ($f \equiv h \; U \; k$) // using the equivalence $h \; U \; k \equiv k \vee (h \wedge X\,(h \; U \; k))$

create two nodes q_1 and q_2 s.t.

$Incoming(q_1) = Incoming(q_2) = Incoming(q)$,

$Old(q_1) = Old(q_2) = Old(q) \cup \{h \; U \; k\}$,

$New(q_1) = New(q) \cup \{h\}$,

$New(q_2) = New(q) \cup \{k\}$,

$Next(q_1) = Next(q) \cup \{h \; U \; k\}$,

$Next(q_2) = Next(q)$;

return expand(q_2, expand(q_1, Nodelist));

else if ($f \equiv h \; R \; k$) // using the equivalence $h \; R \; k \equiv k \wedge (h \vee X\,(h \; R \; k))$

// $\equiv (k \wedge h) \vee (k \wedge X\,(h \; R \; k))$

create two nodes q_1 and q_2 s.t.

$Incoming(q_1) = Incoming(q_2) = Incoming(q)$,

$Old(q_1) = Old(q_2) = Old(q) \cup \{h \; R \; k\}$,

$New(q_1) = New(q) \cup \{h, k\}$,

$New(q_2) = New(q) \cup \{k\}$,

$Next(q_1) = Next(q)$,

$Next(q_2) = Next(q) \cup \{h \; R \; k\}$;

return expand(q_2, expand(q_1, Nodelist));

Completing the Automaton

The resulting Buchi automaton $A = (\Sigma, Q, \Delta, Q_0, F)$

$\Sigma = 2^{AP}$

Q = Nodelist $\cup$ {init}

Q_o = {init}

Δ is defined as follows:

$(q, d, q') \in \Delta$ iff $q \in$ Incoming(q') and
d satisfies the conjunction of negated and unnegated propositions in Old(q')

$F \subseteq 2^Q$ i.e., $F = \{F_1, F_2, \ldots, F_k\}$

The acceptance set F contains a set of accepting states $F_i \in F$ for each subformula of the form h U k where F_i contains all the states q s.t. either $k \in$ Old(q) or h U k $\notin$ Old(q)

If there are no subformulas of the form h U k then F ={Q}

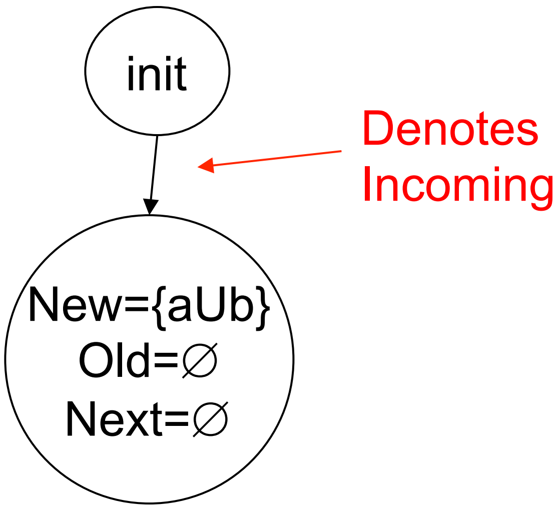
Resulting Automaton

- The size of the resulting automaton can be exponential in the size of the input formula

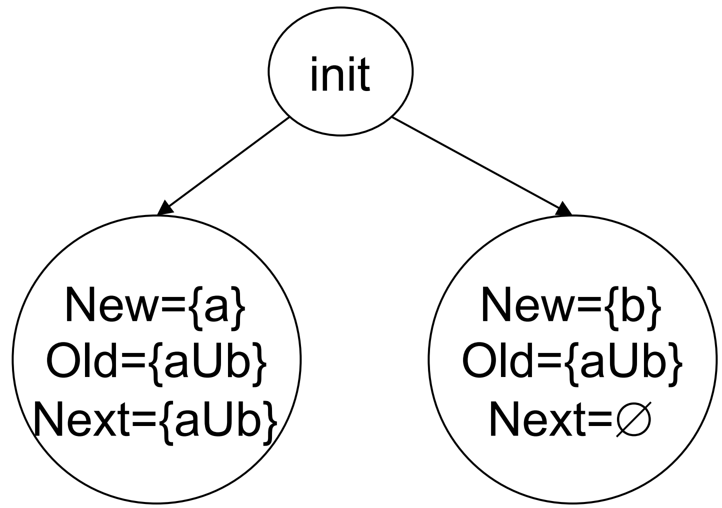
- The resulting automaton is a generalized Buchi automaton
  - we can translate it to a standard Buchi automaton as we discussed earlier

Example Formula: a U b where AP = {a, b}

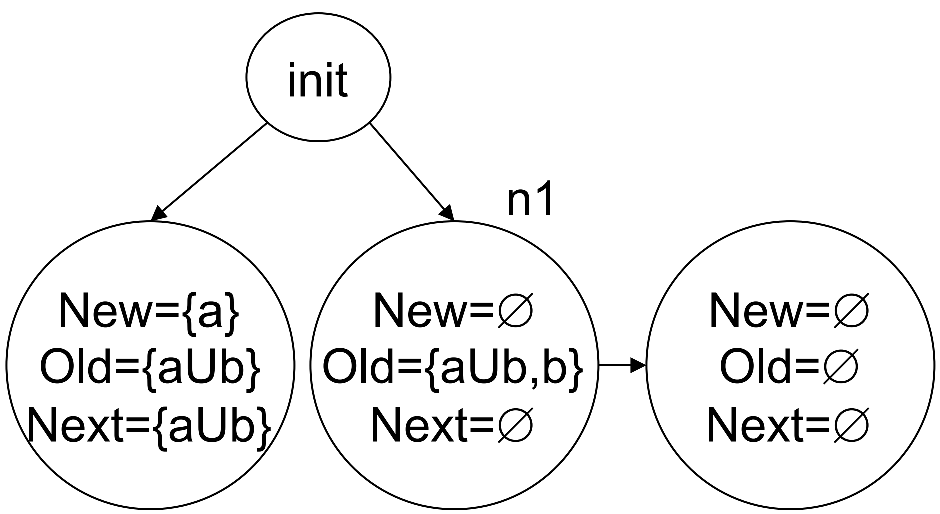
Step 1: Nodelist=∅



Step 2: Nodelist=∅

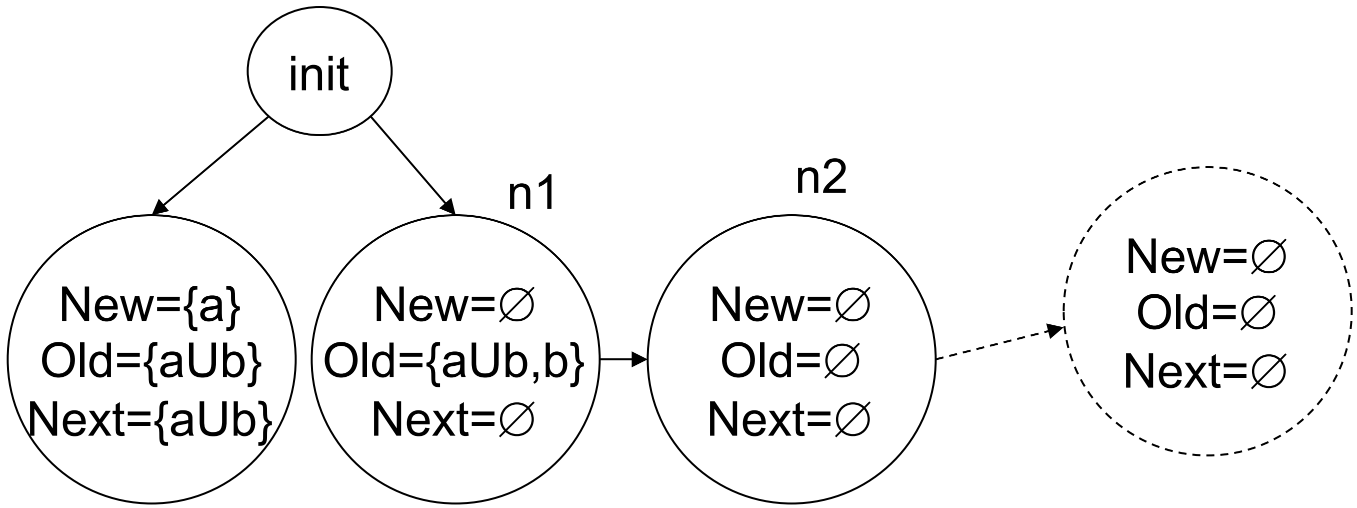


Step 3: Nodelist={n1}

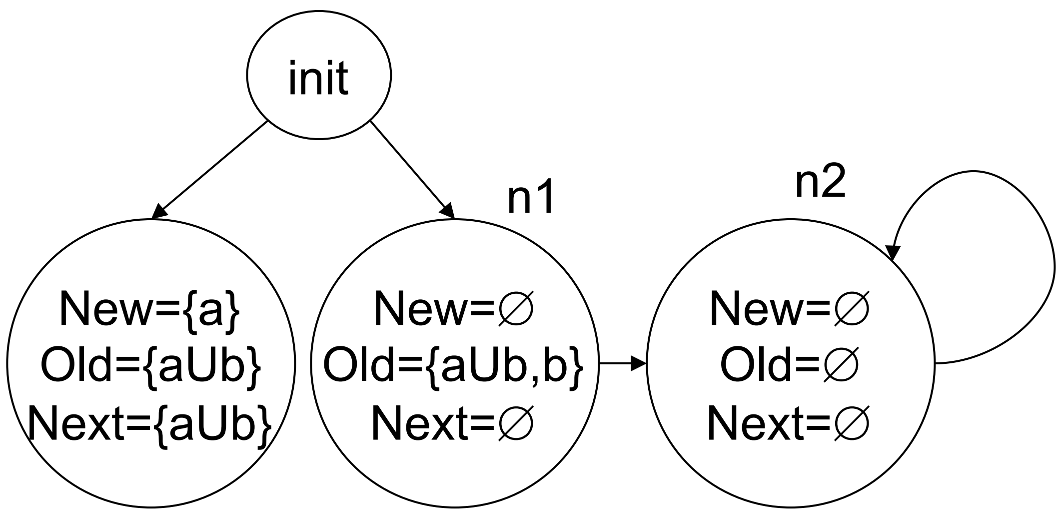


Example (cont’d)

Step 4: Nodelist={n1,n2}

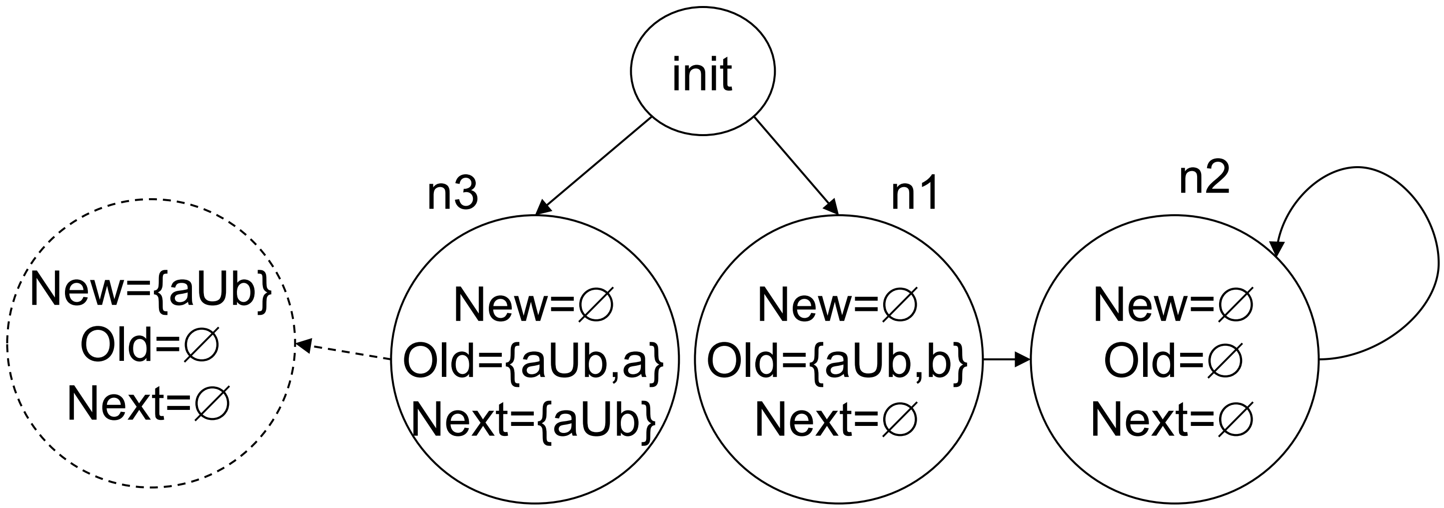


Step 5: Nodelist={n1,n2}

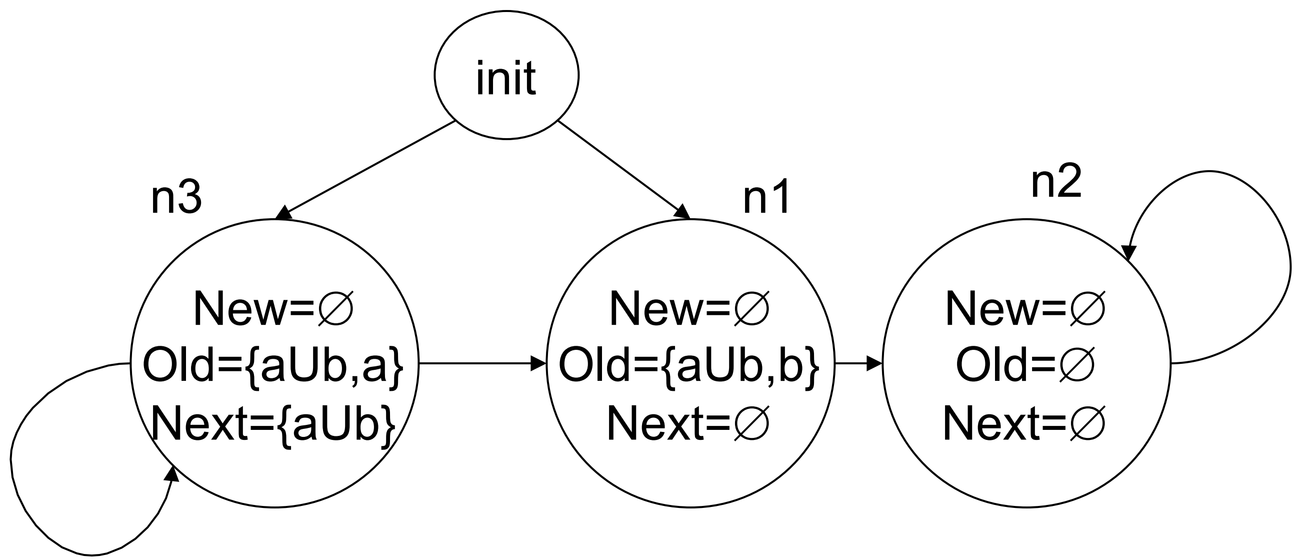


Example (Cont’d)

Step 5: Nodelist={n1,n2,n3}

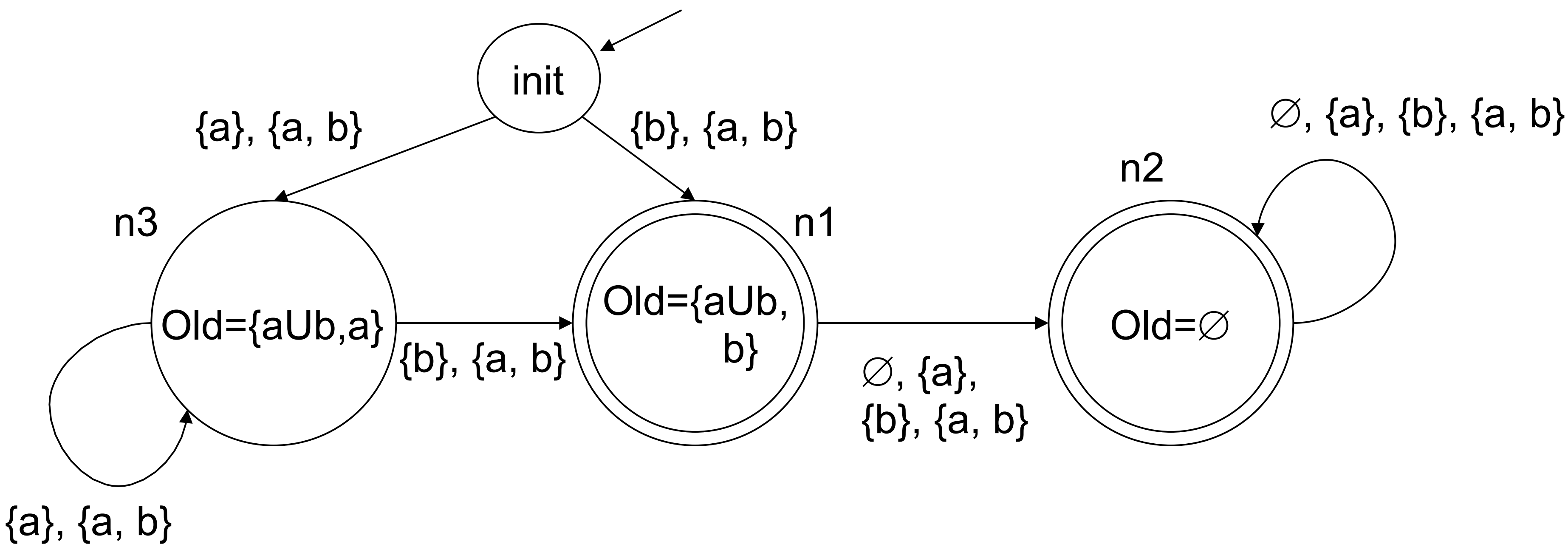


Step 6: Nodelist={n1,n2,n3}



Example (Cont' d)

Final Step: Complete the Automaton



$\Sigma = 2^{AP} = \{ \emptyset, \{a\}, \{b\}, \{a, b\} \}$

$F = \{ \{n1, n2\} \}$

$Q = \{init, n1, n2, n3\}$

$Q_o = \{init\}$