Design and Analysis of Distributed Interacting Systems

Lecture 7 – LTL Model Checking cont.

Prof. Dr. Joel Greenyer

May 30, 2013
Interesting presentations @ ICSE 2013

• E. Letier and W. Heaven, *Requirements Modelling by Synthesis of Deontic Input-Output Automata*  
  (http://www0.cs.ucl.ac.uk/staff/e.letier/publications/2013-ICSE-deonticIOLTS.pdf)

• Problem: Coming up with an adequate formal software specification can be difficult

• Idea: given Domain Assumptions (*Dom*) and Requirements (*Req*), automatically synthesize a software specification (*Spec*), so that *Dom, Spec ⊨ Req* holds.

• Implemented based on LTSA
  – *Dom* and *Req* are given as LTSs with particular shape and interpretation
  – The resulting *Spec* will also be an LTS
Last time: Automata-based LTL Model Checking

- There are different techniques for checking LTL properties
  - i.e. checking whether $M \models \varphi$

- One is based on Büchi Automata (BA)
  - automata that accept infinite words

- Approach: (Be $M$ a Kripke structure over $AP$)

\[
M \models \varphi \\
\text{iff } L(M) \subseteq L(\varphi) \\
\text{iff } L(M) \cap ((2^{AP})^\omega \setminus L(\varphi)) = \emptyset \\
\text{iff } L(M) \cap L(\neg \varphi) = \emptyset \\
\text{iff } L(B_M \otimes B_{\neg \varphi}) = \emptyset
\]

What we need:
1. Checking emptiness of the language accepted by a BA
2. Product construction for BAs
3. Represent KS as BA
4. Represent LTL formula as BA
Agenda

1. Introduce Büchi Automata (√)
2. Checking emptiness of the language accepted by a BA (√)
3. Product construction for BAs (√)
4. Represent KS as BA (√)
5. Represent LTL formula as BA

the slides on LTL model checking are inspired by the model checking lecture of Prof. Heike Wehrheim, University of Paderborn
procedure nested_dfs(BA a)
  forall q_0 \in I_a call dfs_blue(q_0);

procedure dfs_blue (State q)
  q.blue := true;
  forall q' \in post(q) do
    if \neg q'.blue then
      call dfs_blue(q');
    if q \in F_a then
      seed := q;
      call dfs_red(q);

procedure dfs_red (State q)
  q.red := true;
  forall q' \in post(q) do
    if \neg q'.red then
      call dfs_red(q');
    else if q' = seed then
      report cycle;
Last time: Product Construction for BA

- Given two BA $B_1 = (Q_1, \Sigma, T_1, I_1, F_1)$ and $B_2 = (Q_2, \Sigma, T_2, I_2, F_2)$
- Building an automaton $B_1 \otimes B_2$ that accepts $L(B_1) \cap L(B_2)$:
  - $B_1 \otimes B_2 = (Q_1 \times Q_2 \times \{0, 1, 2\}, \Sigma, T, I_1 \times I_2 \times \{0\}, Q_1 \times Q_2 \times \{2\})$
  - we have $((r_i, q_j, x), \sigma, (r_m, q_n, y)) \in T$ iff
    - $(r_i, \sigma, r_j) \in T_1$ and $(q_m, \sigma, q_n) \in T_2$
    - $x = 0$ and $r_m \in F_1$, then $y = 1$
    - $x = 1$ and $q_n \in F_2$, then $y = 2$
    - $x = 2$ then $y = 0$
    - otherwise $x = y$

Represent a Kripke Structure as a Büchi Automaton

• This is quite simple – an example:

Agenda

1. Introduce Büchi Automata (✓)
2. Checking emptiness of the language accepted by a BA (✓)
3. Product construction for BAs (✓)
4. Represent KS as BA (✓)
5. Represent LTL formula as BA

the slides on LTL model checking are inspired by the model checking lecture of Prof. Heike Wehrheim, University of Paderborn
LTL to BA

- There are different ways to construct a BA from LTL formulae
  - declarative construction (not efficient, but easier to understand)
    - see also http://en.wikipedia.org/wiki/Linear_temporal_logic_to_B%C3%BCchi_automaton
  - efficient construction algorithm
    - see also http://en.wikipedia.org/wiki/Linear_temporal_logic_to_B%C3%BCchi_automaton
  - efficient construction algorithm via alternating automata
    - used by LTL2BA library
      - (try web interface: http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/)
- Most approaches first create a Generalized Büchi Automaton (GBA), which is then translated to a regular BA.
Generalized Büchi Automata

• A **Generalized Büchi automaton** is a tuple
  \( GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_k\}) \) in which
  \- \( Q, \Sigma, T, I \) as for regular BAs
  \- \( \{F_1, \ldots, F_k\}, k > 0, F_i \subseteq Q \) is an acceptance condition

• An infinite word \( \pi \in \Sigma^\omega \) is accepted by a GBA iff the GBA has a corresponding run that infinitely often visits at least one state from each set \( F_1, \ldots, F_k \).

• Example:

  \[ F_1 = \{q_1\}, \]
  \[ F_2 = \{q_1, q_2\} \]
GBA to BA (De-Generalization)

• Every GBA can be translated to a regular BA that accepts the same language
• For a GBA = \((Q, \Sigma, T, I, \{F_1, ..., F_n\})\) we construct a BA = \((Q \times \{0, ..., k\}, \Sigma, T', I \times \{0\}, F \times \{k\})\) with 
  \[ ((q, x), \sigma, (q', y)) \in T' \text{ if } (q, \sigma, q') \in T \text{ and } \]
  
  – if \( q \in F_i \) and \( x = i - 1 \) then \( y = i \),
  – if \( x = k \) then \( y = 0 \),
  – \( x = y \) otherwise
GBA to BA (De-Generalization)

• Every GBA can be translated to a regular BA that accepts the same language
• For a \( GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\}) \) we construct a
  \( BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\}) \)
  with
  \[ ((q, x), \sigma, (q', y)) \in T' \text{ if } (q, \sigma, q') \in T \text{ and } \]
  - if \( q \in F_i \) and \( x = i - 1 \) then \( y = i \),
  - if \( x = k \) then \( y = 0 \),
  - \( x = y \) otherwise

• Example:

\[ F_1 = \{q_1\}, \]
\[ F_2 = \{q_1, q_2\} \]
GBA to BA (De-Generalization)

- Every GBA can be translated to a regular BA that accepts the same language.
- For a $GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ we construct a $BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\})$ with $((q, x), \sigma, (q', y)) \in T'$ if $(q, \sigma, q') \in T$ and
  - if $q \in F_i$ and $x = i - 1$ then $y = i$,
  - if $x = k$ then $y = 0$,
  - $x = y$ otherwise.

- Example:

  $F_1 = \{q_1\}$,
  $F_2 = \{q_1, q_2\}$
GBA to BA (De-Generalization)

- Every GBA can be translated to a regular BA that accepts the same language
- For a $GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ we construct a
  $BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\})$
  with $((q, x), \sigma, (q', y)) \in T'$ if $(q, \sigma, q') \in T$ and
    - if $q \in F_i$ and $x = i - 1$ then $y = i$,
    - if $x = k$ then $y = 0$,
    - $x = y$ otherwise
- Example:
  $F_1 = \{q_1\}$,
  $F_2 = \{q_1, q_2\}$
GBA to BA (De-Generalization)

• Every GBA can be translated to a regular BA that accepts the same language

• For a GBA = \((Q, \Sigma, T, I, \{F_1, \ldots, F_n\})\) we construct a BA = \((Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\})\) with 

\(((q, x), \sigma, (q', y)) \in T'\) if \((q, \sigma, q') \in T\) and

- if \(q \in F_i\) and \(x = i - 1\) then \(y = i\),
- if \(x = k\) then \(y = 0\),
- \(x = y\) otherwise

• Example:

\[F_1 = \{q_1\}, \quad F_2 = \{q_1, q_2\}\]
GBA to BA (De-Generalization)

• Every GBA can be translated to a regular BA that accepts the same language

• For a \( GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\}) \) we construct a \( BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\}) \) with \( ((q, x), \sigma, (q', y)) \in T' \) if \( (q, \sigma, q') \in T \) and
  - if \( q \in F_i \) and \( x = i - 1 \) then \( y = i \),
  - if \( x = k \) then \( y = 0 \),
  - \( x = y \) otherwise

• Example:

\[
F_1 = \{q_1\},
F_2 = \{q_1, q_2\}
\]
GBA to BA (De-Generalization)

• Every GBA can be translated to a regular BA that accepts the same language

• For a $GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ we construct a $BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\})$

  with $((q, x), \sigma, (q', y)) \in T'$ if $(q, \sigma, q') \in T$ and

  – if $q \in F_i$ and $x = i - 1$ then $y = i$,
  – if $x = k$ then $y = 0$,
  – $x = y$ otherwise

• Example:

  $F_1 = \{q_1\}$,
  $F_2 = \{q_1, q_2\}$
• Every GBA can be translated to a regular BA that accepts the same language

• For a $GBA = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ we construct a $BA = (Q \times \{0, \ldots, k\}, \Sigma, T', I \times \{0\}, F \times \{k\})$ with $(q, x, \sigma, (q', y)) \in T'$ if $(q, \sigma, q') \in T$ and
  - if $q \in F_i$ and $x = i - 1$ then $y = i$,
  - if $x = k$ then $y = 0$,
  - $x = y$ otherwise

• Example:

$F_1 = \{q_1\}$,
$F_2 = \{q_1, q_2\}$
Declarative Construction – Idea

• Assume that we have an LTL formula $\phi$ with only the basic operators
  – $\neg$
  – $\land$
  – $\mathbf{X}$
  – $\mathbf{U}$
  – (every LTL formula can be converted into such a form)
• Consider all subformulæ of $\phi$, called the closure of $\phi$, $cl(\phi)$

• Example $\phi = a \mathbf{U} (\neg a \land b)$
  – the subformulæ are $a$, $b$, $(\neg a \land b)$, $\phi$,
  together with their negations $\neg a$, $\neg b$, $\neg (\neg a \land b)$, $\neg \phi$
• Be $\varphi$ an LTL formula with basic operators, then $cl(\varphi)$ is inductively defined as follows

- $cl(p) = \{p, \neg p\}, \ p \in AP$
- $cl(\neg \varphi) = cl(\varphi) \cup \{\neg \varphi\}$
- $cl(\varphi_1 \lor \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1 \lor \varphi_2, \neg (\varphi_1 \lor \varphi_2)\}$
- $cl(X \varphi) = cl(\varphi) \cup \{X \varphi, \neg X \varphi\}$
- $cl(\varphi_1 U \varphi_2) = cl(\varphi_1) \cup cl(\varphi_2) \cup \{\varphi_1 U \varphi_2, \neg (\varphi_1 U \varphi_2)\}$

• Maximally consistent subsets of $cl(\varphi)$ are all the biggest subsets $cl(\varphi)$ of that “can hold together” (are not false)

- called $cs(\varphi)$ in the following
- (we skip a constructive definition)
Declarative Construction – Idea cont.

• Consider all *maximal consistent subsets* of $cl(\varphi)$, called $cs(\varphi)$
  
  - these are all maximal combinations of subformulae that “can hold together”, i.e., are not contradicting, i.e., equal to $false$.

• Example $\varphi = a \cup (\neg a \land b)$, $\{a, b\} = AP$
  
  - the subformulae are $a, b, (\neg a \land b), \varphi,$
    together with their negations $\neg a, \neg b, \neg(\neg a \land b), \neg \varphi$
  
  - Maximal consistent sets of subformulae:
    
    \begin{align*}
    \{a, b, \neg(\neg a \land b), \varphi\}, \\
    \{a, b, \neg(\neg a \land b), \neg \varphi\} \\
    \{a, \neg b, \neg(\neg a \land b), \varphi\}, \\
    \{a, \neg b, \neg(\neg a \land b), \neg \varphi\} \\
    \{\neg a, \neg b, \neg(\neg a \land b), \neg \varphi\} \\
    \{\neg a, b, (\neg a \land b), \varphi\}
    \end{align*}
For every run $\pi = a_1, a_2, a_3, \ldots, a_i \subseteq AP$, $\pi \models \varphi$ there exists a sequence of maximally consistent sets of subformulae $\pi' = b_1, b_2, b_3, \ldots$, $b_i \in cs(\varphi)$ such that $b_i \cap AP = a_i$

- we say that $a_i$ is expanded to $b_i$

Example run $\pi = \{a\}, \{a,b\}, \{b\}, \ldots$

- choose $\pi' =$

  $\{a, \neg b, \neg(\neg a \land b), \varphi\}$,
  $\{a, b, \neg(\neg a \land b), \varphi\}$,
  $\{\neg a, b, (\neg a \land b), \varphi\}$, ...

Maximally consistent sets of subformulae:

- $\{a, b, \neg(\neg a \land b), \varphi\}$
- $\{a, b, \neg(\neg a \land b), \neg\varphi\}$
- $\{a, \neg b, \neg(\neg a \land b), \varphi\}$
- $\{a, \neg b, \neg(\neg a \land b), \neg\varphi\}$
- $\{\neg a, \neg b, \neg(\neg a \land b), \neg\varphi\}$
- $\{\neg a, b, (\neg a \land b), \varphi\}$
Furthermore, there especially exists a sequence of maximally consistent sets of subformulae \( \pi' \), such that the following property holds:

- be \( \psi \) a subformula in \( b_i \), \( \psi \in b_i \) iff \( a_i, a_{i+1}, a_{i+2}, \ldots \Vdash \psi \)
- in words: for a particular run, we can choose a sequence of m.c. sets of subformulae, such that for every such set, the remaining corresp. run satisfies the subformulae in it and no other subformulae in \( cl(\varphi) \).

Idea:

- Build GBA with states \( Q = cs(\varphi) \)
- Such that infinite paths starting from a state \( q_i \) accept iff they satisfy every formula in \( q_i \) and refute every formula in \( cl(\varphi) \setminus q_i \)
- start states are such \( q_i \) where \( \varphi \in q_i \)
- ...then the GBA will accept exactly the runs satisfying \( \varphi \)
Declarative Construction – Creating the GBA

• For an LTL formula $\varphi$, we create a
  $\text{GBA} = (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ with

  - $Q = q_0 \cup cs(\varphi)$
  - $I = q_0$
  - $\Sigma = 2^{AP}$
  - $(q_0, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and $\varphi \in q'$
  - $(q, \sigma, q') \in T$ iff ...

Transitions from $q_0$ to states where $\varphi$ holds

$\sigma$ is exactly the set of atomic propositions that are valid in the target state
For an LTL formula $\varphi$, we create a GBA $= (Q, \Sigma, T, I, \{F_1, ..., F_n\})$ with

- $Q = q_0 \cup cs(\varphi)$
- $I = q_0$
- $\Sigma = 2^{AP}$
- $(q_0, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and $\varphi \in q'$
- $(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and
  - $X \psi \in q$ iff $\psi \in q'$$

$\sigma$ is exactly the set of atomic propositions that are valid in the target state

if it says next $\psi$, then $\psi$ should hold next
Declarative Construction – Creating the GBA

• For an LTL formula $\varphi$, we create a GBA $= (Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ with

$- Q = q_0 \cup cs(\varphi)$
$- I = q_0$
$- \Sigma = 2^{AP}$
$- (q_0, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and $\varphi \in q'$
$- (q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and

$- X \psi \in q$ iff $\psi \in q'$, or
$- \psi_1 U \psi_2 \in q$ and $\psi_2 \not\in q$
  then $\psi_1 U \psi_2 \in q'$

from the current state on, we must check whether $\psi_1 U \psi_2$ will be true; but if not yet $\psi_2$ holds, then “remember” that in the next state we must keep checking $\psi_1 U \psi_2$
Declarative Construction – Creating the GBA

• For an LTL formula $\varphi$, we create a GBA = $(Q, \Sigma, T, I, \{F_1, \ldots, F_n\})$ with

  - $Q = q_0 \cup cs(\varphi)$
  - $I = q_0$
  - $\Sigma = 2^{AP}$
  - $(q_0, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and $\varphi \in q'$
  - $(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and
    - $X \psi \in q$ iff $\psi \in q'$, or
    - $\psi_1 U \psi_2 \in q$ and $\psi_2 \not\in q$ then $\psi_1 U \psi_2 \in q'$, or
    - $\psi_1 U \psi_2 \in cl(\varphi) \setminus q$ and $\psi_1 \in q$ then $\psi_1 U \psi_2 \not\in q'$

from the current state on, we must check whether $\psi_1 U \psi_2$ will not be true; if now $\psi_1$ holds, then “remember” to keep checking that it does not turn out that $\psi_1 U \psi_2$ holds
Example

• Example: $\varphi = a \cup b$

• Closure: $cl(\varphi) = \{a, b, \varphi, \neg a, \neg b, \neg \varphi\}$

• Maximal consistent subsets: $cs(\varphi) =$
  
  $q_1 = \{a, b, \varphi\}$
  
  $q_2 = \{a, \neg b, \varphi\}$
  
  $q_3 = \{\neg a, b, \varphi\}$
  
  $q_4 = \{a, \neg b, \neg \varphi\}$
  
  $q_5 = \{\neg a, \neg b, \neg \varphi\}$
Example

- Example: $\varphi = a \cup b$

$(q_0, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and $\varphi \in q'$
Example

• Example: \( \varphi = a \cup b \)

\((q_\varphi, \sigma, q') \in T \iff \sigma = q' \cap AP \text{ and } \varphi \in q' \)
Example: $\phi = a \cup b$

$(q, \sigma, q') \in T \text{ iff } \sigma = q' \cap AP \text{ and...}$

... $\psi_1 \cup \psi_2 \in q \text{ and } \psi_2 \notin q \text{ then } \psi_1 \cup \psi_2 \in q'$
Example

- Example: $\varphi = a \cup b$

$(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and...

... $\psi_1 \cup \psi_2 \in q$ and $\psi_2 \not\in q$ then $\psi_1 \cup \psi_2 \in q'$
Example:

- Example: $\varphi = a \cup b$

$(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and...

... $\psi_1 \cup \psi_2 \in cl(\varphi) \setminus q$ and $\psi_1 \in q$ then $\psi_1 \cup \psi_2 \notin q'$
Example

- Example: $\varphi = a \cup b$

$(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and...

... $\psi_1 \cup \psi_2 \in cl(\varphi) \setminus q$ and $\psi_1 \in q$ then $\psi_1 \cup \psi_2 \not\in q'$
• Example: \( \varphi = a \cup b \)

\[(q, \sigma, q') \in T \text{ iff } \sigma = q' \cap AP \text{ and...}
\]

... <else>

the construction rules do not further restrict the outgoing transitions of these states
• Example: \( \varphi = a \cup b \)

\[ (q, \sigma, q') \in T \iff \sigma = q' \cap AP \text{ and...} \]

… <else>
- Example: $\varphi = a \cup b$

$(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and...

... <else>

we skip the transition labels, they should be clear...
(AP that hold in target state)
• Example: $\varphi = a \cup b$

$(q, \sigma, q') \in T$ iff $\sigma = q' \cap AP$ and...

... <else>

we skip the transition labels, they should be clear...

(AP that hold in target state)
What about the acceptance condition?

- Idea: We have to guarantee that in every state with $\psi_1 \mathbin{U} \psi_2$ we eventually reach a state where $\psi_2$ holds.

- Be $\psi_{1,1} \mathbin{U} \psi_{2,1}$, $\psi_{1,2} \mathbin{U} \psi_{2,2}$, ..., $\psi_{1,k} \mathbin{U} \psi_{2,k}$ all until-formulae in $cl(\phi)$

- then create an acceptance condition $\{F_1, \ldots, F_k\}$ with

  $F_i = \{q \in Q \setminus q_0 \mid \psi_{2,i} \in q \text{ or } \psi_{1,i} \mathbin{U} \psi_{2,i} \not\in q\}$

accepting if $\psi_{2,i}$ holds

or not $\psi_{1,i} \mathbin{U} \psi_{2,i}$ holds
• Example: \( \varphi = a \cup b \)

Here the acceptance condition is \( \{F_1\} \) with

\[
F_1 = \{ q_1, q_3, q_4, q_5 \}
\]
Summary LTL Model Checking, Complexity

- Calculate of $\text{cl}(\varphi)/\text{cs}(\varphi)$: $O(|\varphi|)$, $|\varphi|$ is # of operators in $\varphi$
- Create $B_{\neg \varphi}$: $O(2^{|\varphi|})$
- Create $B_M$: $O(|M|)$, $|M|$ is the number of states and transitions of Kripke Structure M.
- Create $B_M \otimes B_{\neg \varphi}$: $O(|M| \times 2^{|\varphi|})$
- Check $B_M \otimes B_{\neg \varphi}$ emptyness: linear, i.e. $O(|M| \times 2^{|\varphi|})$

$M \models \varphi$

iff $L(M) \subseteq L(\varphi)$

iff $L(M) \cap ((2^{AP})^\omega \setminus L(\varphi)) = \emptyset$

iff $L(M) \cap L(\neg \varphi) = \emptyset$

iff $L(B_M \otimes B_{\neg \varphi}) = \emptyset$

What we need:
1. Checking emptyness of the language accepted by a BA
2. Product construction for BAs
3. Represent KS as BA
4. Represent LTL formula as BA

Hence, overall complexity is linear in size of the model, but exponential in the size of the formula.