Büchi Complementation

Yih-Kuen Tsay
(with help from Chi-Jian Luo)
Department of Information Management
National Taiwan University

FLOLAC 2009
Outline

- Introduction
- Why Is Büchi Complementation Hard?
  - Complementation via Determinization
    - Muller-Schupp Construction
    - Safra’s Construction
    - Safra-Piterman Construction
- Other Approaches
- Concluding Remarks
- References
Introduction

- Languages recognizable by (nondeterministic) Büchi automata are called $\omega$-regular languages.
- The class of $\omega$-regular languages is closed under intersection and complementation (and hence all boolean operations).
- Deterministic Büchi automata are strictly less expressive.
- The complement of a deterministic Büchi automaton may not be deterministic.
While intersection is rather straightforward, complementation is much harder and still a current research topic.

A complementation construction is also useful for checking **language containment** (and hence equivalence) between two automata:

$$L(A) \subseteq L(B) \equiv L(A) \cap L(B^c) = \phi.$$  

The language containment test is essential in the **automata-theoretic approach** to model checking (more about this later ...).
Complementation of an NFA

- Translate the given nondeterministic finite automaton (NFA) $N$ into an equivalent deterministic finite automaton (DFA) $D$ via the subset construction.
- Take the dual of $D$ to get a DFA $D'$ for the complement language.
- This works because languages recognizable by DFA’s are closed under complementation.
Example of NFA Complementation

- \( L(N) = (a+b)^*aa^* \), which equals \((a+b)^*a\).

NFA \( N \)

- An equivalent DFA \( D \) by the subset construction.

DFA \( D \)

There are two unreachable states in \( D \).
Subset Construction for Finite Words

Formally, from NFA $N=(S_N, \Sigma, \delta_N, q_0, F_N)$, we construct an equivalent DFA $D=(S_D, \Sigma, \delta_D, \{q_0\}, F_D)$ as follows:

- $S_D = 2^{S_N}$
- $\delta_D(S, a) = \bigcup_{s \in S} \delta_N(s, a)$
- $F_D = \{ S \in S_D \mid S \cap F_N \neq \phi \}$
ω-Automata

- **ω-automata** are finite automata on infinite words.
- Büchi automata are one type of ω-automata.
- Formally, a (nondeterministic) ω-automaton $B$ is represented as a five-tuple $B=(\Sigma, S, s_0, \delta, \text{Acc})$:
  - $\Sigma$: a finite alphabet (set of symbols)
  - $S$: a finite set of states (or locations)
  - $s_0 \in S$: the initial state
  - $\delta: S \times \Sigma \rightarrow 2^S$
  - $\text{Acc}$: the acceptance condition

When $\delta$ is actually a function from $S \times \Sigma$ to $S$, the automaton is said to be *deterministic*. 
A run of an \( \omega \)-automaton \( B \) on a word \( w = w_1w_2 \ldots \) is an infinite sequence of states \( s_0s_1 \ldots \in S^\omega \) such that for all \( j \geq 0 \) we have \( s_{j+1} \in \delta(s_j, w_{j+1}) \).

For a run \( r \), let \( \text{Inf}(r) \) denote the set of states that occur infinitely many times in \( r \).

A word \( w \) is accepted by \( B \) if there exists an accepting run of \( B \) on \( w \) that satisfies the acceptance condition.

The language of \( B \), denoted \( L(B) \), is the set of all words accepted by \( B \).
Büchi and Other $\omega$-Automata

- **Büchi automata:**
  
  $\text{Acc} = F \subseteq S$.

  A run $r$ is accepting iff $\text{Inf}(r) \cap F \neq \emptyset$.

- **Parity automata:**
  
  $\text{Acc} = \{F_0, F_1, \ldots, F_k\}, F_i \subseteq S$.

  A run $r$ is accepting iff the smallest $i$ such that $\text{Inf}(r) \cap F_i \neq \emptyset$ is even.
Büchi and Other $\omega$-Automata (cont.)

- **Rabin automata:**

  \[ Acc = \{(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)\}, \ E_i, F_i \subseteq S. \]

  A run $r$ is accepting iff for some $i$, $\text{Inf}(r) \cap E_i = \emptyset$ and $\text{Inf}(r) \cap F_i \neq \emptyset$.

- **Streett automata:**

  \[ Acc = \{(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)\}, \ E_i, F_i \subseteq S. \]

  A run $r$ is accepting iff for all $i$, $\text{Inf}(r) \cap E_i \neq \emptyset$ or $\text{Inf}(r) \cap F_i = \emptyset$.

- **Rabin automata and Streett automata** are the dual of each other.
Convenient Acronyms

- DBW (or DBA): deterministic Büchi automata
- NBW: nondeterministic Büchi automata
- DPW: deterministic parity automata
- DRW: deterministic Rabin automata
- DSW: deterministic Streett automata
- etc.

Note: replace W with T, for tree automata.
An Example of Büchi Automaton

- $B = (\{a, b\}, \{q_0, q_1\}, \{q_0\}, T, \{q_1\})$
  - $T(q_0,a) = \{q_0, q_1\}$
  - $T(q_0,b) = \{q_0\}$
  - $T(q_1,a) = \{q_1\}$
  - $T(q_1,b) = \{\}$

- Apparently, $B$ is nondeterministic.

- $L(B) = (a+b)^*a^\omega$ (or “FG a” or “<>[a]”).
If we use the subset construction to construct a DBW $D$ from an NBW $N$, the two automata may not be language equivalent.

By construction, the accepting states of the DBW $D$ are those that contain an accepting state of the original NBW $N$.

$D$ may accept some words that are rejected by $N$, as shown by the following example.

Thus, this method is not sound.
Naive Subset Construction

- NBW $N$ defines the language: $(a+b)^*a^\omega$ ("eventually always $a$").

![Diagram of NBW N]

- $N$ accepts words like $ababa^\omega$ and $bbba^\omega$.
- $N$ rejects words like $(ab)^\omega$ and $bb(ba)^\omega$.

A DBW $D$ by the naive subset construction.

![Diagram of DBW D]

(unreachable states removed)

- $D$ accepts every word that is accepted by $N$.
- However, $D$ also accepts some words that are rejected by $N$, e.g., $(ab)^\omega$. 
Another Subset Construction

- This subset construction keeps more detailed information of accepting states visited in a run.
- A state of $D$ is called a **breakpoint** if the state does not contain any unmark state of $N$.
- The construction will mark an accepting state of $N$ and every state that has a marked predecessor.
- A word $w$ is accepted if $D$ identifies **infinitely many breakpoints** while reading $w$.
- This does not work, either; see the example next.
Another Subset Construction (cont.)

- This automaton accepts the input word $a^\omega$.
- The constructed automaton also has a run on $a^\omega$, which is accepting.
Another Subset Construction (cont.)

- This automaton also accepts the input word $b^\omega$.

- However, the single run of the constructed automaton on $b^\omega$ is rejecting:

$$\begin{align*}
{s0} & \xrightarrow{b} \{s0, s2!\} & \xrightarrow{b} \{s0, s2!\} & \xrightarrow{b} \{s0, s2!\} & \xrightarrow{b} \ldots
\end{align*}$$

- Therefore, this construction is incomplete, missing words that should be accepted.
Duality Does Not Apply

If we take the dual of a given DBW $D$ to get DBW $D'$, then it is possible that $L(D) \cap L(D') \neq \emptyset$, e.g., $(ab)^\omega$.

Note: DBW is not closed under complementation, e.g., $((a+b)*a)^\omega$ (or GF $a$).
Muller-Schupp Construction

- We shall now study three constructions for Büchi complementation.

- Stages in Muller-Schupp construction:
  - NBW $\rightarrow$ DRW $\rightarrow$ (complete) DSW $\rightarrow$ NBW
  - The DSW is the complement of the DRW, by taking the dual view.

- The determinization part uses Muller-Schupp trees to construct the DRW.

- A Muller-Schupp tree (MS tree) is a finite strictly binary tree, which has precisely two children for each node except the leave nodes.
Run Trees vs. Run DAG’s

In Figure (a) is an example run tree $r_w$ and in (b) is the corresponding run DAG $r_d$. 
MS Trees

- In a run tree $r_w$, we partition the children of a node $v$ into two classes, the left child which carries an accepting state and the right one which carries a non-accepting state.
- Let us refer to the new tree as $t_1$.
- Claim: $r_w$ has an accepting path iff $t_1$ has a path branching left infinitely often.
MS Trees (cont.)

- For every state \( s \) on each level in \( t_1 \), if we only keep the leftmost \( s \), we obtain another new tree \( t_2 \).
- Claim: \( t_1 \) has a path branching left infinitely often iff \( t_2 \) has a path branching left infinitely often.
MS Trees (cont.)

\[
\begin{align*}
&\text{w1} & & \{s0\} \\
&\text{w2} & & \{s0, s3\} \\
&\text{w3} & & \{s2\}
\end{align*}
\]

\[
\begin{align*}
&\text{w1} & & \{s1\} \\
&\text{w2} & & \{s4\} \\
&\text{w3} & & \{s5\}
\end{align*}
\]

\[
\begin{align*}
&\text{w1} & & \{s0\} \\
&\text{w2} & & \{s1, s3\} \\
&\text{w3} & & \{s2\}
\end{align*}
\]

\[
\begin{align*}
&\text{w1} & & \{s1, s2, s3\} \\
&\text{w2} & & \{s1, s4, s5\} \\
&\text{w3} & & \{s2\}
\end{align*}
\]

1\{s0\}r
2\{s1, s3\}g
3\{s2\}r
1\{s1, s4, s5\}r
2\{s4, s5\}y
5\{s4\}r
6\{s1\}g
1\{s1, s2, s5\}r
2\{s1, s2\}y
3\{s5\}g
4\{s1\}g
5\{s2\}r
6\{s5\}g

2009/7/9 Yih-Kuen Tsay

FLOLAC 2009: Büchi Complementation
Three Colors for the Nodes

Three colors are used to identify whether a node is accepting or not.

- A node is red if the run path that the node represents has no accepting state.
- A node is yellow if it has visited an accepting state before but it does not visit an accepting state in this step.
- A node is green if it visits an accepting state in this step or it merges a green or yellow son.
An Example of MS Construction

2009/7/9  Yih-Kuen Tsay

FOLAC 2009: Büchi Complementation
An Example of MS Construction (cont.)

compute successors

\[ 1\{q_0,q_3\} \text{r} \]

\[ 2\{y\} \quad 3\{q_0,q_3\} \text{r} \]

\[ 5\{q_3\} \text{y} \quad 6\{q_0\} \text{r} \]

create sons

\[ 1\{q_0,q_3\} \text{r} \]

\[ 2\{y\} \quad 3\{q_0,q_3\} \text{r} \]

\[ 5\{q_3\} \text{y} \quad 6\{q_0\} \text{r} \]

\[ 4\{q_3\} \text{g} \quad 7\{q_0\} \text{r} \]

remove empty

\[ 1\{q_0,q_3\} \text{r} \]

\[ 3\{q_0,q_3\} \text{r} \]

\[ 5\{q_3\} \text{y} \quad 6\{q_0\} \text{r} \]

\[ 4\{q_3\} \text{g} \quad 7\{q_0\} \text{r} \]

merge sons

\[ 1\{q_0,q_3\} \text{r} \]

\[ 5\{q_3\} \text{g} \quad 6\{q_0\} \text{r} \]

\[ b \]

2009/7/9 Yih-Kuen Tsay

FLOLAC 2009: Büchi Complementation

27 / 53
An Example of Rejecting a Word

200/7/9 Yih-Kuen Tsay

FOLAC 2009: Büchi Complementation
The Detail of Determinization

Let $A = (\Sigma, S, s_0, \delta, F')$ be an NBW with $n$ states.

An equivalent DRW $D = (\Sigma, S', s_0', \delta', Acc)$:

- $S'$: a set of MS trees,
- $s_0'$: an initial MS tree with only one node numbered 1, which is labeled $\{s_0\}$ and colored red,
- $\delta'$: a transition function which, given an input $a \in \Sigma$, transforms an MS tree using the steps described next.
- $Acc = \{(E_1,F_1), (E_2,F_2), \ldots, (E_{4n},F_{4n})\}$:
  - $E_i$: the set of MS trees without node $i$.
  - $F_i$: the set of MS trees with green node $i$. 
Steps to compute the next MS-tree state:

- Change color green to yellow for every tree node.
- Replace the label of every node with $\bigcup_{s \in L} \delta(s, a)$.
- Create a left child with label $L \cap F$ and a right child with label $L \setminus F$.
- Merge the same states into the leftmost one for each level in the tree.
- Remove every node with an empty label.
- Mark green every node that has only one child with color green or yellow.
Safra’s Construction

- Stages of the complementation:
  - NBW $\rightarrow$ DRW $\rightarrow$ (complement) DSW $\rightarrow$ NBW
- Safra trees are used to construct the DRW.
- Safra trees are labeled ordered trees.
Safra Trees

FLOLAC 2009: Büchi Complementation
An Example of Construction

1\{q0\}

compute successors

1\{q0, q1\}

create sons

1\{q0, q1\}

2\{q1\}

compute successors

2\{q1\}

3\{q1\}

merge states

1\{q0, q1\}

4\{q1\}

remove empty

1\{q0, q1\}

2\{q1\}

merge sons

1\{q0, q1\}

2\{q1\}!!

a

a

2\{q1\}

3\{q1\}

4\{q1\}
An Example of Construction (cont.)

2007/7/9 Yih-Kuen Tsay

FOLAC 2009: Büchi Complementation
An Example of Rejecting a Word

2009/7/9  Yih-Kuen Tsay  FLOLAC 2009: Büchi Complementation  35 / 53
An Example of Rejecting a Word

create sons

1\{q0, q1\}
2\{q1\}
3\{q1\}
4\{q1\}

compute successors

1\{q0, q1\}
2\{q1\}
3\{q1\}
4\{q1\}

2\{\}
4\{\}
1\{q0\}
3\{\}

remove empty

1\{q0\}

compute successors

1\{q0, q1\}
2\{q1\}

create sons

1\{q0, q1\}
2\{q1\}

merge states

1\{q0, q1\}
2\{q1\}
3\{\}
4\{q1\}

remove empty

1\{q0, q1\}
2\{q1\}

merge sons

1\{q0, q1\}
2\{q1\}!
Detail of the Determinization

Let $A = (\Sigma, S, s_0, \delta, F)$ be an NBW with $n$ states.

An equivalent DRW $D = (\Sigma, S', s_0', \delta', \text{Acc}')$:

- $S'$: a set of Safra trees,
- $s_0'$: an initial Safra tree with only one node numbered 1 which is labeled $\{s_0\}$,
- $\delta'$: a transition function which, given an input $a \in \Sigma$, transforms a Safra tree using the steps described next,
- $\text{Acc}' = \{(E_1,F_1),(E_2,F_2), \ldots, (E_{2n},F_{2n})\}$:
  - $E_i$ = the set of Safra trees without node $i$.
  - $F_i$ = the set of Safra trees with marked node $i$. 

37FLOLAC 2009: Büchi Complementation
Detail of the Determinization (cont.)

Steps to compute the next Safra-tree state:

- Remove the mark of every tree node.
- Create a new child with label $L \cap F$.
- Replace the label of every node with $\bigcup_{s \in L} \delta(s, a)$.
- Merge the same states into the leftmost one for each level in the tree.
- Remove every node with an empty label.
- Mark every node whose label equals the union of the labels of its children and remove its children.
Safra-Piterman Construction

- Stages of the complementation:
  - NBW $\rightarrow$ DPW $\rightarrow$ (complement) DPW $\rightarrow$ NBW
- The determinization part uses compact Safra trees to construct the DPW.
- Compact Safra trees are Safra trees, but use two different kinds of techniques:
  - Dynamic names
  - Recording only the smallest marked name (called $f$) and removed name (called $e$)
Dynamic Names

- The construction renames the tree at the final step and get a new tree.
- But it does not change the marks of the smallest $e$ and $f$. 

Diagram:

- Some step
  - e=2, f=3
    - $1\{L1\}$
    - $3\{L2\}!$
    - $4\{L3\}$
  - rename
    - $1\{L1\}$
    - $2\{L2\}!$
    - $3\{L3\}$
  - $w_i$
  - Some step
    - e=4, f=2
    - $1\{L1\}$
    - $2\{L2\}!$
    - $3\{L3\}$
An Example of Construction

compute successors

create sons

merge states

compute successors

create sons

merge sons

s0

s2

s3

\[
es = 2, f = 1
\]

\[
es = 3, f = 3
\]

\[
es = 4, f = 2
\]

\[
es = 3, f = 3
\]

\[
es = 3, f = 3
\]

\[
es = 4, f = 2
\]

\[
es = 4, f = 2
\]

\[
es = 4, f = 2
\]

\[
es = 4, f = 2
\]

\[
es = 4, f = 2
\]
An Example of Construction (cont.)

- **completing successors**
  - vertex $v$:
    - $e=4$, $f=2$
    - create sons
      - $1\{q0,q3\}$
      - $3\{q3\}$
      - $4\{q3\}$

- **merging sons**
  - $e=4$, $f=2$
  - remove empty
    - $1\{q0,q3\}$
    - $3\{q3\}!$

- **renaming**
  - $e=2$, $f=3$
  - $s4$
  - $b$

- **completing successors**
  - $e=2$, $f=3$
  - $2\{q3\}$

- **merging sons**
  - $e=2$, $f=3$
  - $1\{q0,q3\}$
  - $2\{q3\}$
  - $3\{q3\}$
The Determinization

- Let $A = (\Sigma, S, s_0, \delta, F)$ be an NBW with $n$ states.
- An equivalent DPW $D = (\Sigma, S', s_0', \delta', Acc')$:
  - $S'$: the set of compact Safra trees,
  - $s_0'$: an initial compact Safra tree with only one node numbered 1, which is labeled $\{s_0\}$ and has $e=2$ and $f=1$,
  - $\delta'$: a transition function which, given an input $a \in \Sigma$, transforms a compact Safra tree as described next,
  - The acceptance condition $Acc' = \{F_0, F_1, ..., F_{4n}\}$:
    - $F_0 = \{s \in S' | f = 1\}$.
    - $F_{2i+1} = \{s \in S' | e = i+2$ and $f \geq e\}$.
    - $F_{2i+2} = \{s \in S' | f = i+2$ and $e > f\}$.
    - $i = \{0, 1, 2, ..., 2n-1\}$.
The Determinization (cont.)

Steps to compute the next compact Safra-tree state:

- Replace the label of every node with $\bigcup_{s \in L} \delta(s, a)$.
- Create a new child with label $L \cap F$.
- Merge the same states into the leftmost one for each level in the tree.
- For every node, whose label equals the union of the labels of its children, remove its children and assign the smallest number of these nodes to $f$.
- Remove every node with an empty label and set $e$ to the smallest number of removed node.
Comparison

- We define a modified Safra’s construction, which is similar to the original one, except that we exchange the step of computing successors and the step of creating children.
- Let us compare these four algorithms: Safra, modified Safra, Safra-Piterman, Muller-Schupp.
Comparison (cont.)

input word: $aaa(b)^\omega$

Safra

\[
\begin{align*}
1\{q0\} & \xrightarrow{a} 1\{q0,q1,q2\} \\
& \xrightarrow{a} 1\{q0,q1,q2,q3\} \\
& \xrightarrow{a} 2\{q1\} \\
& \xrightarrow{a} 2\{q1\} \\
& \xrightarrow{a} 3\{q3\}
\end{align*}
\]

Modified Safra

\[
\begin{align*}
1\{q0\} & \xrightarrow{a} 1\{q0,q1,q2\} \\
& \xrightarrow{a} 2\{q1\} \\
& \xrightarrow{a} 1\{q0,q1,q2,q3\} \\
& \xrightarrow{a} 2\{q1\} \\
& \xrightarrow{a} 3\{q3\}
\end{align*}
\]

Piterman

\[
\begin{align*}
e=2, f=1 \\
1\{q0\} & \xrightarrow{a} 1\{q0,q1,q2\} \\
& \xrightarrow{a} 2\{q1\} \\
e=3, f=3 \\
1\{q0,q1,q2\} & \xrightarrow{a} 1\{q0,q1,q2,q3\} \\
& \xrightarrow{a} 2\{q1\} \\
& \xrightarrow{a} 3\{q3\}
\end{align*}
\]

Muller-Schupp

\[
\begin{align*}
e=2, f=2 \\
1\{q0\}r & \xrightarrow{a} 1\{q0,q1,q2\}r \\
& \xrightarrow{a} 2\{q1\}r \\
& \xrightarrow{a} 3\{q0,q2\}r \\
e=4, f=2 \\
1\{q0,q1,q2,q3\}r & \xrightarrow{a} 1\{q0,q1,q2,q3\}r \\
& \xrightarrow{a} 2\{q1\}r \\
& \xrightarrow{a} 3\{q0,q2,q3\}r \\
& \xrightarrow{a} 5\{q3\}r \\
& \xrightarrow{a} 6\{q0,q2\}r
\end{align*}
\]
Comparison (cont.)

input word: $aaa(b)^\omega$

\[
\begin{align*}
&\text{e}=4, \ e=2, \ f=3, \ f=2
\end{align*}
\]
Some Observations

- Modified Safra trees are slightly better than Safra trees, because a modified Safra tree is usually one step ahead of the corresponding Safra tree.
- Safra-Piterman trees are usually better than modified Safra trees, because a Safra-Piterman tree only cares about the smallest marked name in the tree.
- Modified Safra trees are sometimes better than Safra-Piterman trees, because the rename step spends some time and adds some states.
Some Observations (cont.)

- Muller-Schupp trees are the largest, because they contain more redundant data.
- Safra-Piterman construction performs better than others, because DPW can be translated into NBW more efficiently.
- Muller-Schupp construction helps to understand other algorithms.
Other Complementation Algorithms

- [Thomas]
  - NBW $\rightarrow$ APW $\rightarrow$ (complement) NBW
    - APW: alternating parity automaton

- [Kupferman and Vardi]
  - NBW $\rightarrow$ (complement) UCBW $\rightarrow$ VWAA $\rightarrow$ NBW
    - UCBW: universal co-Büchi automaton
    - VWAA: very weak alternating automaton

There is also a construction (by Kurshan) for DBW complementation, which is quite efficient.
Concluding Remarks

- Büchi complementation is expensive.
- The automata-theoretic approach to model checking tries to avoid it:
  - The system is modeled as a Büchi automaton $A$.
  - A desired property is given by a PTL formula $f$.
  - Let $B_f$ ($B_{\sim f}$) denote a Büchi automaton equivalent to $f$ ($\sim f$).
  - The model checking problem translates into
    \[ L(A) \subseteq L(B_f) \text{ or } L(A) \cap L(B_{\sim f}) = \emptyset \text{ or } L(A \times B_{\sim f}) = \emptyset. \]
  - So, with PTL to automata translation, the expensive complementation procedure is avoided.
- The well-used model checker SPIN, for example, adopts the automata-theoretic approach and asks the user to express properties in LTL.
Concluding Remarks (cont’d)

- When the B in $A \subseteq B$ is given by an arbitrary Büchi automaton, complementation cannot be avoided.
- However, complementation of B may be done “on demand”.
- When the containment does not hold, one might find a counterexample before going through the full procedure of complementation.
- There are algorithms for checking language containment based on this idea.
- This line of research is still ongoing.
References