Symbolic algorithms for language equivalence and KAT

Damien Pous
Plume group, CNRS, ENS Lyon
PoPL, Mumbai, 16.01.2015
Language equivalence of automata

• Useful for
  - Model checking
  - Program equivalence (cf. D’Antoni’s tutorial)
  - SDN analysis (NetKAT, cf. previous talk)
  - Formal proof automation (demo at the end of the talk)
Language equivalence of automata

• Useful for
  − Model checking
  − Program equivalence (cf. D’Antoni’s tutorial)
  − SDN analysis (NetKAT, cf. previous talk)
  − Formal proof automation (demo at the end of the talk)

• This work:
  − Symbolic version of Hopcroft&Karp’s algorithm
  − Application to Kleene algebra with tests (KAT)
Hopcroft & Karp’s algorithm ('71)

\[ x \xrightarrow{a,b} y \xrightarrow{a,b} z \xleftarrow{a,b} \]

\[ u \xrightarrow{a} v \xleftarrow{a,b} w \xrightarrow{a,b} \]

- Use a union-find datastructure to represent equivalence classes
- Actually an instance of Huet’s unification algorithm ('76)
- Almost linear in \(|S| \cdot |A|\) (\(S\): states, \(A\): alphabet)
Hopcroft & Karp’s algorithm (’71)

Use a union-find datastructure to represent equivalence classes

Actually an instance of Huet’s unification algorithm (’76)

Almost linear in $|S| \cdot |A|$ (\(S\): states, \(A\): alphabet)
Hopcroft & Karp’s algorithm (’71)

- Use a union-find datastructure to represent equivalence classes
- Actually an instance of Huet’s unification algorithm (’76)
- Almost linear in $|S| \cdot |A|$ ($S$: states, $A$: alphabet)
Hopcroft & Karp’s algorithm (’71)

• Use a union-find datastructure to represent equivalence classes
• Actually an instance of Huet's unification algorithm (’76)
• Almost linear in \(|S| \cdot |A|\)

(S: states, A: alphabet)
Hopcroft & Karp’s algorithm (’71)

- Use a union-find datastructure to represent equivalence classes
- Actually an instance of Huet’s unification algorithm (’76)
- Almost linear in $|S| \cdot |A|$

\( S \): states, \( A \): alphabet
Hopcroft & Karp’s algorithm ('71)

- Use a union-find datastructure to represent equivalence classes
- Actually an instance of Huet’s unification algorithm ('76)
- Almost linear in $|S| \cdot |A|$ (S: states, A: alphabet)
Hopcroft & Karp’s algorithm (’71)

• Use a union-find datastructure to represent equivalence classes
• Actually an instance of Huet’s unification algorithm (’76)
• Almost linear in $|S| \cdot |A|$ (S: states, A: alphabet)
Hopcroft & Karp’s algorithm (’71)

- Use a **union-find** datastructure to represent equivalence classes

\[ \begin{align*}
X & \xrightarrow{a,b} y \xrightarrow{a,b} z \xleftarrow{a,b} w \\
\bar{v} & \xrightarrow{a,b} \bar{v} \\
u & \xrightarrow{a} \xleftarrow{b} \}
\]
Hopcroft & Karp’s algorithm (’71)

- Use a union-find datastructure to represent equivalence classes
Hopcroft & Karp’s algorithm ('71)

- Use a union-find datastructure to represent equivalence classes
- Actually an instance of Huet’s unification algorithm ('76)
- Almost linear in $|S| \cdot |A| \quad (S: \text{states}, \ A: \text{alphabet})$
Few states, many letters

\[
\begin{array}{c}
X \\
\uparrow \\
V \\
\uparrow \\
U \\
\downarrow a \\
W \\
\end{array}
\begin{array}{c}
\rightarrow a,b \\
\rightarrow a,b \\
\rightarrow a,b \\
\rightarrow a,b \\
\leftarrow a,b \\
\uparrow a \leftarrow b \\
\end{array}
\begin{array}{c}
Y \\
\rightarrow a,b \\
\rightarrow a,b \\
\rightarrow a,b \\
\leftarrow a,b \\
\downarrow a,b \\
\end{array}
\begin{array}{c}
\rightarrow a,b \\
\rightarrow a,b \\
\rightarrow a,b \\
\leftarrow a,b \\
\downarrow a,b \\
\end{array}
\begin{array}{c}
Z \\
\rightarrow a,b \\
\leftarrow a,b \\
\downarrow a,b \\
\end{array}
\begin{array}{c}
\rightarrow a,b \\
\rightarrow a,b \\
\rightarrow a,b \\
\leftarrow a,b \\
\downarrow a,b \\
\end{array}
\begin{array}{c}
W \\
\end{array}
\]
Few states, many letters

- Standard practice: label transitions with formulas
- But then Hopcroft & Karp's algorithm can no longer be used

Damien Pous, CNRS, ENS Lyon
Few states, many letters

• standard practice: label transitions with formulas
Few states, many letters

- standard practice: label transitions with formulas
Few states, many letters

- standard practice: label transitions with formulas
  but then Hopcroft & Karp’s algorithm can no longer be used
Few states, many letters

- standard practice: label transitions with formulas
  but then Hopcroft&Karp’s algorithm can no longer be used
Few states, many letters

- standard practice: label transitions with formulas
  but then Hopcroft&Karp’s algorithm can no longer be used
- more restrictive model here
Binary Decision Diagrams (BDDs) [Bryant’86]

Represent functions of type $2^A \rightarrow V$ with compressed decision trees

**Example:** $A=\{a,b\}, \; V=\{v,w\}, \; \alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$
Binary Decision Diagrams (BDDs) [Bryant’86]

Represent functions of type $2^A \rightarrow V$ with compressed decision trees

Example: $A=\{a,b\}$, $V=\{v,w\}$, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$
Binary Decision Diagrams (BDDs) [Bryant’86]

Represent functions of type $2^A \rightarrow V$ with compressed decision trees

Example: $A=\{a,b\}$, $V=\{v,w\}$, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$
Binary Decision Diagrams (BDDs) [Bryant’86]

Represent functions of type $2^A \rightarrow V$ with compressed decision trees

Example: $A=\{a,b\}$, $V=\{v,w\}$, $\alpha \mapsto \begin{cases} v & \text{si } \alpha(a) = 1 \text{ et } \alpha(b) = 0 \\ w & \text{sinon} \end{cases}$
Symbolic Automata

Just represent the transition functions with BDDs

Example:
alphabet: $2^A$ with $A = \{a, b, c\}$
output set: $\mathbb{N}$
Iterate over the successors of two states
Iterate over the successors of two states

We just need an “iter2” function on BDDs
iter2 on BDDs
Symbolic algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Symbolic Hopcroft & Karp’s algorithm

The union-find datastructure can be used between BDD nodes!
Kleene algebra with tests
Kleene algebra with tests [Kozen’97]

\[ \text{KAT} = \text{Kleene algebra} + \text{Boolean algebra} \]
Kleene algebra with tests [Kozen’97]

\[ \text{KAT} = \text{Kleene algebra} + \text{Boolean algebra} \]

\[ p, q \in \Sigma \]

\[ p + (p^* \cdot q)^* \]
Kleene algebra with tests [Kozen’97]

$\text{KAT} = \text{Kleene algebra} + \text{Boolean algebra}$

$p, q \in \Sigma$

$p + (p^* \cdot q)^*$

$a, b \in A$

$a \land \neg b \lor \neg a$
Kleene algebra with tests [Kozen’97]

\[
\text{KAT} = \text{Kleene algebra } + \text{ Boolean algebra } \\
p, q \in \Sigma \\
p + (p^* \cdot q)^* \\
\]

\[
a, b \in A \\
a \land !b \lor !a \\
a \cdot !b + !a
\]
Kleene algebra with tests [Kozen’97]

\[
\text{KAT} = \text{Kleene algebra} + \text{Boolean algebra}
\]

\[
p, q \in \Sigma \\
p + (p^* \cdot q)^*
\]

\[
a, b \in A \\
a \land !b \lor !a \\
a \cdot !b + !a
\]

\[
a \cdot p + !a \cdot q
\]

\[
(b \cdot p)^* \cdot !b
\]
Kleene algebra with tests [Kozen’97]

\[ KAT = \text{Kleene algebra} + \text{Boolean algebra} \]

\[ p, q \in \Sigma \]

\[ a, b \in A \]

\[ p + (p^* \cdot q)^* \]

\[ a \land \neg b \lor \neg a \]

\[ a \land \neg b + \neg a \]

\[ a \cdot p + \neg a \cdot q \]

\[ (b \cdot p)^* \cdot \neg b \]

Automata model: alphabet \( 2^A \times \Sigma \), output set \( 2^A \rightarrow 2 \)
In the paper

Symbolic automata constructions for KAT:

- Brzozowski’s derivatives
- Antimirov’ partial derivatives
- Ilie & Yu’s construction
- determinisation
- $\epsilon$-removal

Only using natural and efficient BDD operations
http://perso.ens-lyon.fr/damien.pous/symbolickat/
http://perso.ens-lyon.fr/damien.pous/symbolickat/

http://perso.ens-lyon.fr/damien.pous/ra/

[ITP’13]
To remember

• Symbolic version of Hopcroft & Karp’s algorithm → mixing BDDs and Union-Find

• Efficient algorithms for KAT → to be extended to NetKAT

http://perso.ens-lyon.fr/damien.pous/symbolickat/