Testing Equivalence of NFAs using Congruence

Martin Hruška[1], Ondřej Lengál[1,2], and Tomáš Vojnar[1]

[1]Brno University of Technology, Czech Republic
[2]Academia Sinica, Taiwan

Motivation

Language equivalence of NFAs:

- A general problem with many applications.
- Language-preserving reduction of NFAs:
  - if two states have the same language, we can merge them.
- Testing equal behaviour of systems:
  - e.g. proving correctness of optimisations.
- Testing termination criteria of fixpoint computations (ARMC).
- Can be used to decide language inclusion (shown later).

Based on the following papers:

- J. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. TR 114, Cornell Univ. 1971.
- F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. POPL'13.

Testing Language Equivalence of NFAs

- Can be done by minimization:
  - Make a disjoint union of the input automata.
  - Determinize and minimize.
  - If the original initial states end up in the same equivalence class, the automata have equal languages.
  - In fact, equivalence between each pair of states is checked this way.

Testing Language Equivalence of NFAs

- Can be done by minimization:
  - Make a disjoint union of the input automata.
  - Determinize and minimize.
  - If the original initial states end up in the same equivalence class, the automata have equal languages.
  - In fact, equivalence between each pair of states is checked this way.

- One can also try to build a bisimulation on the disjoint union of the determinized automata relating the initial states.
  - Start with the relation consisting of the pair of the initial states of the two automata only.
  - Try to iteratively add pairs to create a bisimulation.
  - No need to determinize beforehand, can be done on the fly, parts of the implicit subset construction may be avoided as shown later on.

Foundation of the Bisimulation-based Approach

The theoretical foundation of the bisimulation-based approach is given by the following lemma:

Lemma

Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be an NFA and $X, Y \subseteq Q$. Then

$$\mathcal{L}(X) = \mathcal{L}(Y) \quad \textit{iff} \quad (X \cap F \neq \emptyset \Leftrightarrow Y \cap F \neq \emptyset) \ \wedge \ \forall a \in \Sigma \,.\, \mathcal{L}(Post_a(X)) = \mathcal{L}(Post_a(Y)).$$

Proof.

$\Rightarrow$ easy, by contradiction

$\Leftarrow$ easy, by construction □

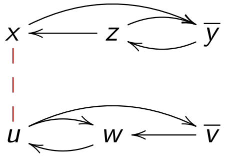
Basic Algorithm

Algorithm: `LangEquiv`(X, Y)
$R := \emptyset$; $todo := \emptyset$;
$todo$.`insert`$((X, Y))$;
while $todo \neq \emptyset$ **do**
    $(X', Y') := todo$.`get_and_remove`();
    if $(X', Y') \in R$ **then continue**;   `// this will change`
    if $(X' \cap F \neq \emptyset) \not\Leftrightarrow (Y' \cap F \neq \emptyset)$ **then return** `false` ;
    foreach $a \in \Sigma$ **do**
        $todo$.`insert`$((Post_a(X'), Post_a(Y')))$;
    R.`insert`$((X', Y'))$;
return `true`;

- A direct implementation of the lemma;
- if no counterexample is reachable, $\mathcal{L}(X) = \mathcal{L}(Y)$.

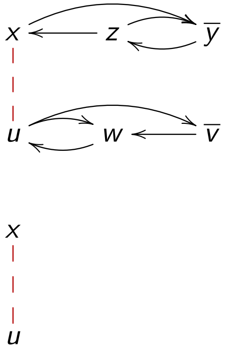
Checking language equivalence

Non-deterministic case: use Hopcroft and Karp on the fly:



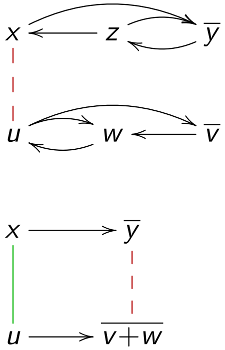
Checking language equivalence

Non-deterministic case: use Hopcroft and Karp on the fly:



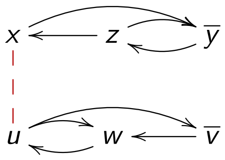
Checking language equivalence

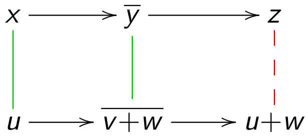
Non-deterministic case: use Hopcroft and Karp on the fly:



Checking language equivalence

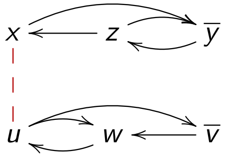
Non-deterministic case: use Hopcroft and Karp on the fly:

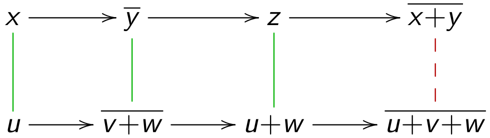




Checking language equivalence

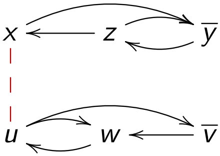
Non-deterministic case: use Hopcroft and Karp on the fly:

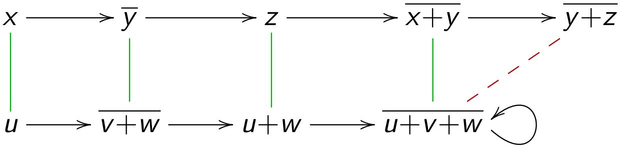




Checking language equivalence

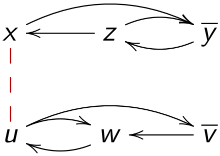
Non-deterministic case: use Hopcroft and Karp on the fly:

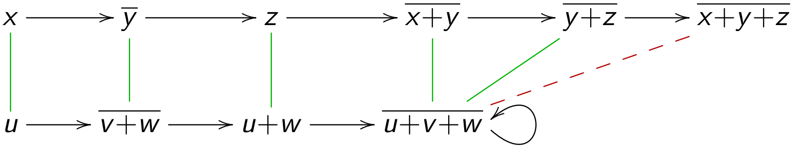




Checking language equivalence

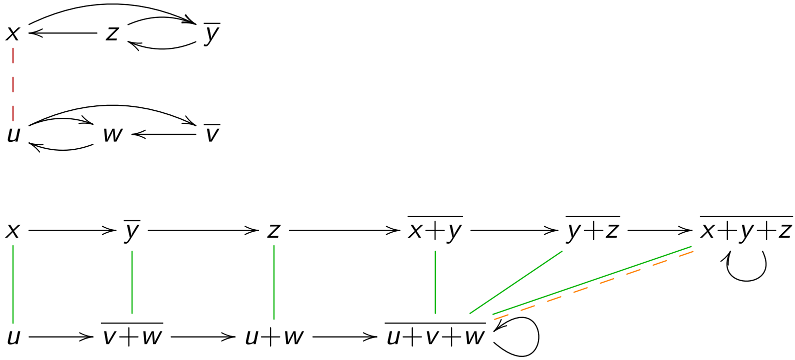
Non-deterministic case: use Hopcroft and Karp on the fly:





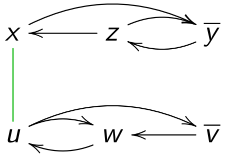
Checking language equivalence

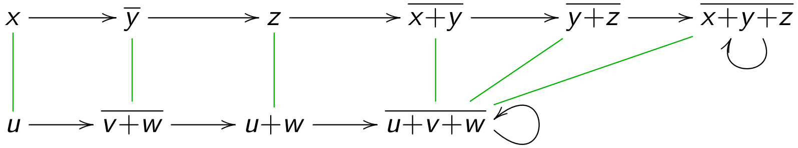
Non-deterministic case: use Hopcroft and Karp on the fly:



Checking language equivalence

Non-deterministic case: use Hopcroft and Karp on the fly:





Optimisation 1: Equivalence (Hopcroft and Karp '71)

Algorithm: `LangEquiv`(X, Y)

$R := \emptyset$; $todo := \emptyset$;

$todo$.`insert`$((X, Y))$;

while $todo \neq \emptyset$ **do**

  $(X', Y') := todo$.`get_and_remove`$()$;

  if $(X', Y') \in e(R \cup todo)$ **then continue**; // `equivalence`

  if $(X' \cap F \neq \emptyset) \not\Leftrightarrow (Y' \cap F \neq \emptyset)$ **then return** `false` ;

  foreach $a \in \Sigma$ **do**

    $todo$.`insert`$((Post_a(X'), Post_a(Y')))$;

  R.`insert`$((X', Y'))$;

return `true`;

- $e(\varrho)$ is the reflexive, symmetric, and transitive closure of ϱ.
- Rationale:
  - $\mathcal{L}(X') = \mathcal{L}(X')$,
  - $\mathcal{L}(X') = \mathcal{L}(Y') \implies \mathcal{L}(Y') = \mathcal{L}(X')$, and
  - $\mathcal{L}(X') = \mathcal{L}(\alpha) \land \mathcal{L}(\alpha) = \mathcal{L}(Y') \implies \mathcal{L}(X') = \mathcal{L}(Y')$.

Optimisation 2: Congruence (Bonchi and Pous '13)

Algorithm: $\texttt{LangEquiv}(X, Y)$

$R := \emptyset$; $todo := \emptyset$;

$todo.\texttt{insert}((X, Y))$;

while $todo \neq \emptyset$ **do**

  $(X', Y') := todo.\texttt{get\_and\_remove}()$;

  if $(X', Y') \in c(R \cup todo)$ **then continue**; // $e()$ + congruence

  if $(X' \cap F \neq \emptyset) \not\Leftrightarrow (Y' \cap F \neq \emptyset)$ **then return** false ;

  foreach $a \in \Sigma$ **do**

    $todo.\texttt{insert}((Post_a(X'), Post_a(Y')))$;

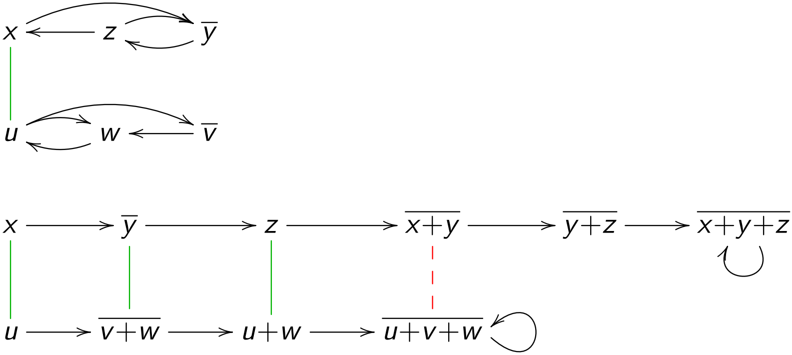
  $R.\texttt{insert}((X', Y'))$;

return true;

- $c(\varrho)$ is the congruence closure of $e(\varrho)$ w.r.t. $[\,\cdot \cup \cdot\,]$:
  - $(\alpha, \alpha') \in c(\varrho) \wedge (\beta, \beta') \in c(\varrho) \implies (\alpha \cup \beta, \alpha' \cup \beta') \in c(\varrho)$
- Rationale:
  - $\mathcal{L}(\alpha') = \mathcal{L}(\alpha') \wedge \mathcal{L}(\beta) = \mathcal{L}(\beta') \implies \mathcal{L}(\alpha \cup \beta) = \mathcal{L}(\alpha' \cup \beta')$

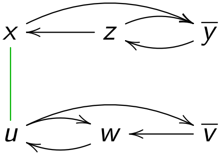
Checking language equivalence

One can do better:

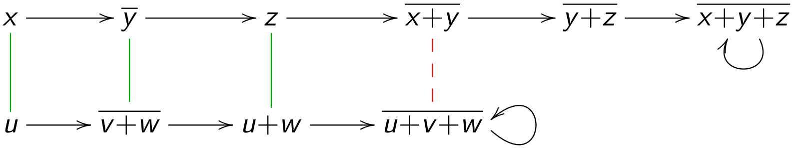


Checking language equivalence

One can do better:

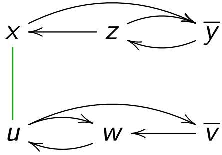


$$\begin{array}{rl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$

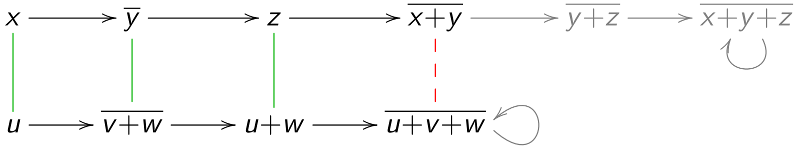


Checking language equivalence

One can do better:



$$\begin{array}{cl} & (x,\ u) \\ + & (y,\ v{+}w) \\ \hline = & (x{+}y,\ u{+}v{+}w) \end{array}$$



parts of the accessible subsets need not be explored

Testing $(X', Y') \in c(\varrho)$

- Eagerly computing $c(\varrho)$ is infeasible.
  - Note that $c(\varrho) \subseteq 2^Q \times 2^Q \quad \rightsquigarrow \quad$ can get exponential!

- Testing $(X', Y') \in c(\varrho)$ efficiently:
  - Saturate both X' and Y' as follows:
    - for $Z \subseteq X'$:

      $$\text{if } (Z, W) \in (\varrho \cup \varrho^{-1}), \text{ rewrite } X' \rightsquigarrow X' \cup W;$$

    - keep repeating for both sides until a fixpoint (X'^F, Y'^F).
  - Test whether $X'^F = Y'^F$.
  - Time complexity: $\mathcal{O}(|\varrho|^2 \cdot |Q|)$,
    - Search the pairs in ϱ and states to find an applicable pair; repeat – but each pair can be applied once only.

An Example of the Saturation

$x + y$

u

$$R = \{(x, u), (y + z, u)\}$$

An Example of the Saturation

$x + y$ $\qquad\qquad\qquad\qquad$ u

$x + y + u$

$$R = \{(x, u), (y + z, u)\}$$

An Example of the Saturation

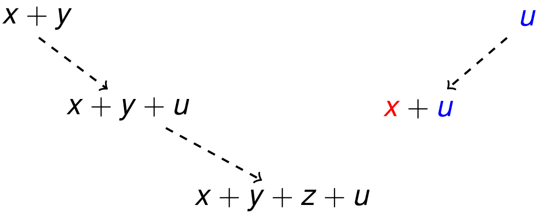
$x + y$ $\qquad\qquad\qquad\qquad$ u

$x + y + u$

$x + y + z + u$

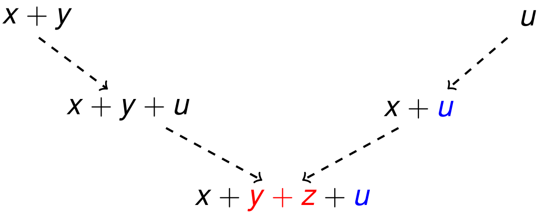
$$R = \{(x, u), (y + z, u)\}$$

An Example of the Saturation



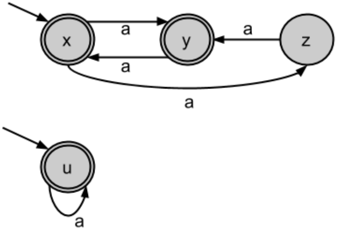
$$R = \{(x, u), (y + z, u)\}$$

An Example of the Saturation

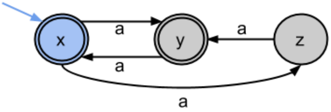


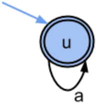
$$R = \{(x, u), (y + z, u)\}$$

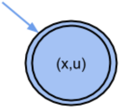
A Complete Example of Equivalence Checking



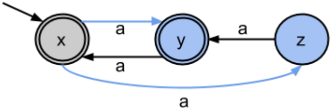
A Complete Example of Equivalence Checking

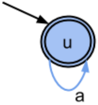


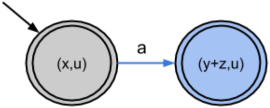




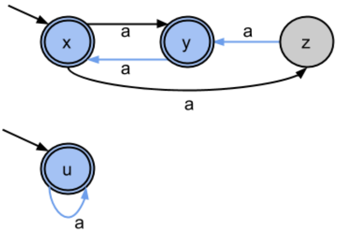
A Complete Example of Equivalence Checking

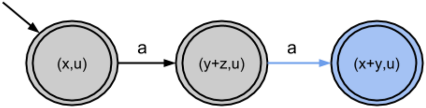




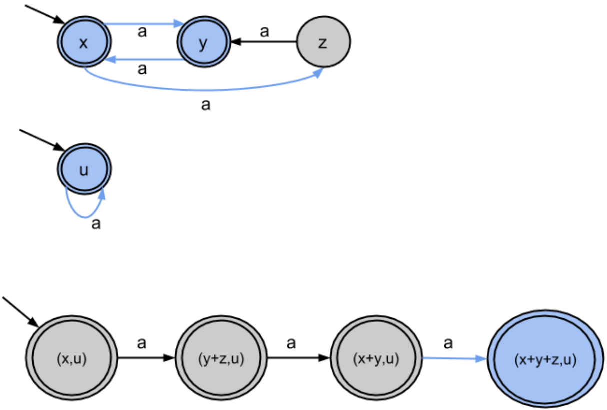


A Complete Example of Equivalence Checking

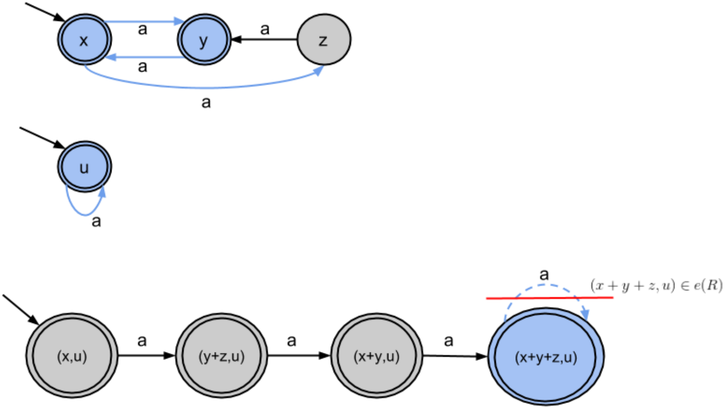




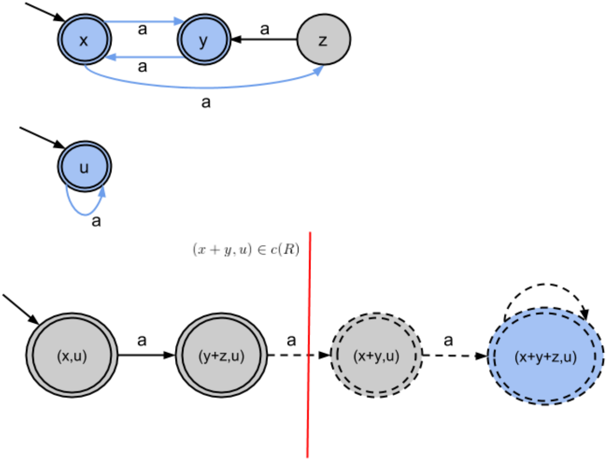
A Complete Example of Equivalence Checking



A Complete Example of Equivalence Checking



A Complete Example of Equivalence Checking



Optimisation 3: Simulations (Bonchi and Pous '13)

```
Algorithm: LangEquiv(X, Y)
R := ∅; todo := ∅;
todo.insert((X, Y));
while todo ≠ ∅ do
    (X′, Y′) := todo.get_and_remove();
    if (X′, Y′) ∈ cs(R ∪ todo) then continue;        // c() + sim.
    if (X′ ∩ F ≠ ∅) ⇎ (Y′ ∩ F ≠ ∅) then return false ;
    foreach a ∈ Σ do
        todo.insert((Post_a(X′), Post_a(Y′)));
    R.insert((X′, Y′));
return true;
```

- $cs(\varrho)$ is $c(\varrho \cup \{(\{x\}, \{x, y\}) \mid y \preceq x\})$,
- $\preceq$ is an under-approximation of language inclusion,
  - e.g. forward simulation.

An Example: Congruences and Simulations (1)

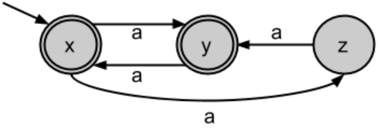


Table : Simulations can be computed separately for the given automata, e.g., for the above NFA out of the two sooner considered, we get:

$\preceq$	x	y	z
x	1	1	0
y	1	1	0
z	1	1	1

For the other considered (single state) NFA, the simulation is trivial.

Saturation with Simulation: An Example

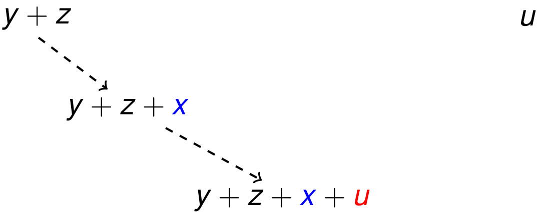
$y + z$ $\qquad$ u

$$R = \{(x, u)\}$$

Saturation with Simulation: An Example

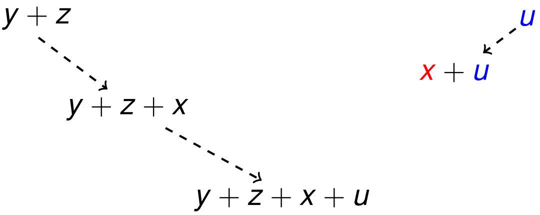
$$R = \{(x, u)\}, \; x \preceq y$$

Saturation with Simulation: An Example



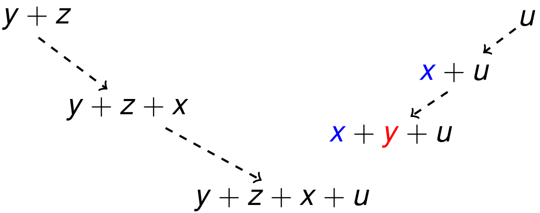
$$R = \{(x, u)\}$$

Saturation with Simulation: An Example



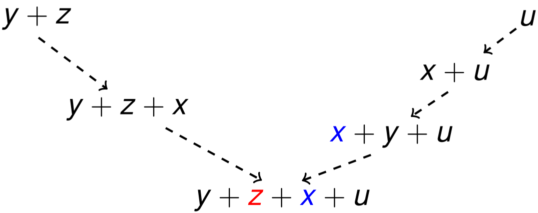
$$R = \{(x, u)\}$$

Saturation with Simulation: An Example



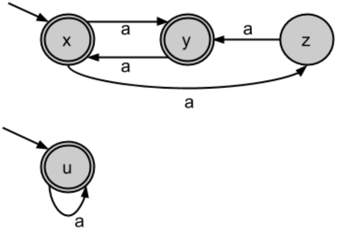
$$R = \{(x, u)\}, \ y \preceq x$$

Saturation with Simulation: An Example

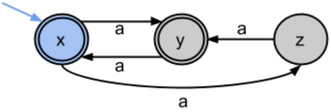


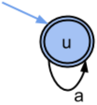
$$R = \{(x, u)\}, z \preceq x$$

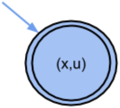
An Example: Congruences and Simulations (2)



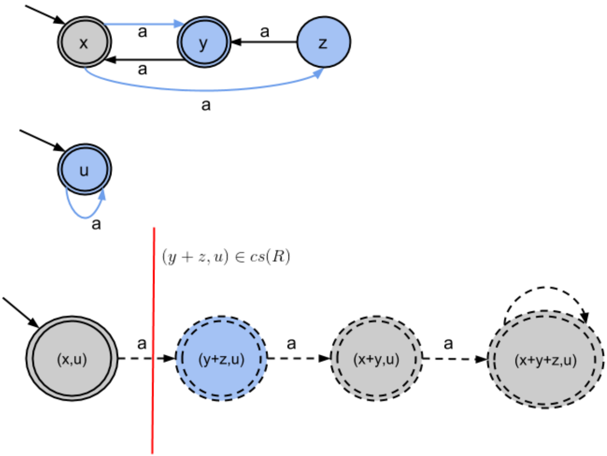
An Example: Congruences and Simulations (2)







An Example: Congruences and Simulations (2)



An Example: Congruences and Simulations (3)

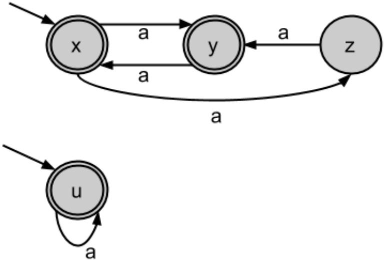


Table : Simulation can also be computed over the disjoint union of both of the given automata:

$\preceq$	x	y	z	u
x	1	1	0	1
y	1	1	0	1
z	1	1	1	1
u	1	1	0	1

Saturation with Simulation: Another Example

x

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x + y$$

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x + y \xrightarrow{z \preceq x} x + y + z$$

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x+y \xrightarrow{z \preceq x} x+y+z \xrightarrow{u \preceq x} x+y+z+u = X'$$

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x+y \xrightarrow{z \preceq x} x+y+z \xrightarrow{u \preceq x} x+y+z+u = X'$$

$$u \xrightarrow{x \preceq u} x + u$$

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x+y \xrightarrow{z \preceq x} x+y+z \xrightarrow{u \preceq x} x+y+z+u = X'$$

$$u \xrightarrow{x \preceq u} x + u \xrightarrow{y \preceq u} x + y + u$$

Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x+y \xrightarrow{z \preceq x} x+y+z \xrightarrow{u \preceq x} x+y+z+u = X'$$

$$u \xrightarrow{x \preceq u} x+u \xrightarrow{y \preceq u} x+y+u \xrightarrow{z \preceq u} x+y+z+u = Y'$$

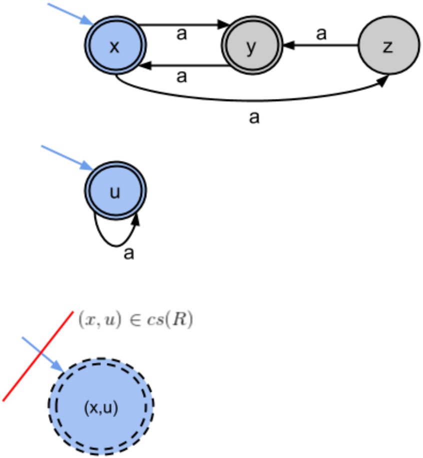
Saturation with Simulation: Another Example

$$x \xrightarrow{y \preceq x} x+y \xrightarrow{z \preceq x} x+y+z \xrightarrow{u \preceq x} x+y+z+u = X'$$

$$u \xrightarrow{x \preceq u} x+u \xrightarrow{y \preceq u} x+y+u \xrightarrow{z \preceq u} x+y+z+u = Y'$$

$$X' = Y' \Rightarrow (x, u) \in cs(R)$$

An Example: Congruences and Simulations (4)



Use for Inclusion Checking

Observe the following:

$$\mathcal{L}(A) \subseteq \mathcal{L}(B) \iff \mathcal{L}(A) \cup \mathcal{L}(B) = \mathcal{L}(B)$$

Structure of pairs:

$$(X_A \cup Y_B, Y_B)$$

Possible optimisation:

$$(X_A \cup Y_B, Y_B) \in c(\varrho) \iff X_A \subseteq Y_B^F$$

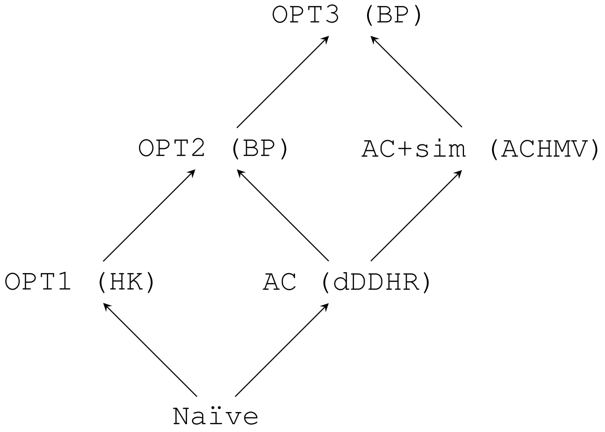
An example:

$$\varrho = \{(\{x, u\}, \{u\}), (\{y, z, u\}, \{u\})\}$$

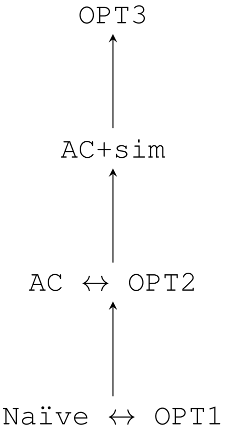
$$(\{x, y, u\}, \{u\}) \overset{?}{\in} c(\varrho)$$

$$\{u\} \xrightarrow{(\{x,u\},\{u\})\in\varrho} \{x, u\} \xrightarrow{(\{y,z,u\},\{u\})\in\varrho} \{x, y, z, u\} \quad \supseteq \quad \{x, y, u\}$$

Congruence vs. Antichains ($x \to y$: y can mimick x)



Single NFA, equivalence



Pair of NFAs, inclusion

Congruence vs. Antichains in Inclusion Checking (1)

Consider checking inclusion $\mathcal{L}(A) \overset{?}{\subseteq} \mathcal{L}(B)$, i.e. $\mathcal{L}(A) \cup \mathcal{L}(B) \overset{?}{=} \mathcal{L}(B)$:

- The pairs are of the form $(X_A \cup Y_B, Y_B)$.
- HK does not work: transitivity and symmetry do not apply.

Congruence vs. Antichains in Inclusion Checking (1)

Consider checking inclusion $\mathcal{L}(A) \stackrel{?}{\subseteq} \mathcal{L}(B)$, i.e. $\mathcal{L}(A) \cup \mathcal{L}(B) \stackrel{?}{=} \mathcal{L}(B)$:

- The pairs are of the form $(X_A \cup Y_B, Y_B)$.
- HK does not work: transitivity and symmetry do not apply.
- `AC` $\rightarrow$ `OPT2`:
  - Let $(\{x_A\} \cup Y_B, Y_B) \in R$, i.e. $(x_A, Y_B) \in AC$.
  - `AC` can discard any (x_A, Y'_B) where $Y'_B \supseteq Y_B$.
  - `OPT2` can mimic this:
    - $Y'_B = Y_B \cup Z$ for some Z,
    - $Y'_B \quad = \quad Y_B \cup Z \quad \stackrel{R}{\rightsquigarrow} \quad \{x_A\} \cup Y_B \cup Z \quad = \quad \{x_A\} \cup Y'_B$, and so
    - $(\{x_A\} \cup Y'_B, Y'_B) \in c(R)$.

Congruence vs. Antichains in Inclusion Checking (1)

Consider checking inclusion $\mathcal{L}(A) \overset{?}{\subseteq} \mathcal{L}(B)$, i.e. $\mathcal{L}(A) \cup \mathcal{L}(B) \overset{?}{=} \mathcal{L}(B)$:

- The pairs are of the form $(X_A \cup Y_B, Y_B)$.
- HK does not work: transitivity and symmetry do not apply.
- `AC` $\to$ `OPT2`:
  - Let $(\{x_A\} \cup Y_B, Y_B) \in R$, i.e. $(x_A, Y_B) \in AC$.
  - `AC` can discard any (x_A, Y'_B) where $Y'_B \supseteq Y_B$.
  - `OPT2` can mimic this:
    - $Y'_B = Y_B \cup Z$ for some Z,
    - $Y'_B \quad = \quad Y_B \cup Z \quad \overset{R}{\rightsquigarrow} \quad \{x_A\} \cup Y_B \cup Z \quad = \quad \{x_A\} \cup Y'_B$, and so
    - $(\{x_A\} \cup Y'_B, Y'_B) \in c(R)$.
- `OPT2` $\to$ `AC`:
  - Based on $(\{1, a, b\}, \{a, b\}), (\{2, b, c\}, \{b, c\}) \in R$, `OPT2` can discard $(\{1, 2, a, b, c\}, \{a, b, c\})$.
  - `AC` can mimic this:
    - with $(1, \{a, b\}), (2, \{b, c\}) \in AC$,
    - both $(1, \{a, b, c\})$ and $(2, \{a, b, c\})$ can be discarded.

Congruence vs. Antichains in Inclusion Checking (2)

- $\texttt{AC} + \texttt{sim} \rightarrow \texttt{OPT3}$:
  - Consider
    - antichain $AC = \{(x_1, \{y_2\}), (x_2, \{y_1, y_3\})\}$
    - simulation $\{x_1 \preceq x_1, y_1 \preceq y_1, y_1 \preceq x_1\}$,
    - and a pair $(x_2, \{y_2, y_3\})$.

Congruence vs. Antichains in Inclusion Checking (2)

- $\texttt{AC} + \texttt{sim} \to \texttt{OPT3}$:
  - Consider
    - antichain $AC = \{(x_1, \{y_2\}), (x_2, \{y_1, y_3\})\}$
    - simulation $\{x_1 \preceq x_1, y_1 \preceq y_1, y_1 \preceq x_1\}$,
    - and a pair $(x_2, \{y_2, y_3\})$.
  - $\texttt{AC} + \texttt{sim}$ cannot discard $(x_2, \{y_2, y_3\})$,

Congruence vs. Antichains in Inclusion Checking (2)

- $\mathtt{AC} + \mathtt{sim} \rightarrow \mathtt{OPT3}$:
  - Consider
    - antichain $AC = \{(x_1, \{y_2\}), (x_2, \{y_1, y_3\})\}$
    - simulation $\{x_1 \preceq x_1, y_1 \preceq y_1, y_1 \preceq x_1\}$,
    - and a pair $(x_2, \{y_2, y_3\})$.
  - $\mathtt{AC} + \mathtt{sim}$ cannot discard $(x_2, \{y_2, y_3\})$,
  - $\mathtt{OPT3}$ can discard $(\{x_2, y_2, y_3\}, \{y_2, y_3\})$:

$$\{y_2, y_3\} \xrightarrow{(\{x_1, y_2\}, \{y_2\}) \in R} \{x_1, y_2, y_3\}$$

$$\xrightarrow{y_1 \preceq x_1} \{x_1, y_1, y_2, y_3\}$$

$$\xrightarrow{(\{x_2, y_1, y_3\}, \{y_1, y_3\}) \in R} \{x_1, x_2, y_1, y_2, y_3\} \supseteq \{x_2, y_2, y_3\}$$

Congruence vs. Antichains in Inclusion Checking (2)

- $\texttt{AC} + \texttt{sim} \rightarrow \texttt{OPT3}$:
  - Consider
    - antichain $AC = \{(x_1, \{y_2\}), (x_2, \{y_1, y_3\})\}$
    - simulation $\{x_1 \preceq x_1, y_1 \preceq y_1, y_1 \preceq x_1\}$,
    - and a pair $(x_2, \{y_2, y_3\})$.
  - $\texttt{AC} + \texttt{sim}$ cannot discard $(x_2, \{y_2, y_3\})$,
  - $\texttt{OPT3}$ can discard $(\{x_2, y_2, y_3\}, \{y_2, y_3\})$:

$$\{y_2, y_3\} \xrightarrow[]{(\{x_1, y_2\}, \{y_2\}) \in R} \{x_1, y_2, y_3\}$$

$$\xrightarrow[]{y_1 \preceq x_1} \{x_1, y_1, y_2, y_3\}$$

$$\xrightarrow[]{(\{x_2, y_1, y_3\}, \{y_1, y_3\}) \in R} \{x_1, x_2, y_1, y_2, y_3\} \supseteq \{x_2, y_2, y_3\}$$

  - Unlike $\texttt{AC} + \texttt{sim}$, $\texttt{OPT3}$ can combine pairs from R and $\preceq$.