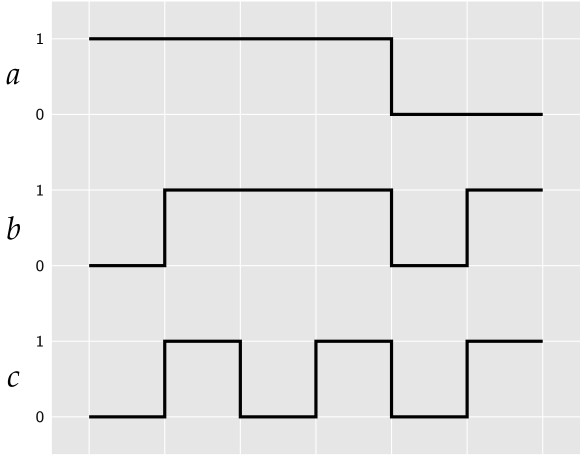
Translation of Semi-Extended Regular Expressions using Derivatives

Antoine Martin[1], Etienne Renault[2], Alexandre Duret-Lutz[1]

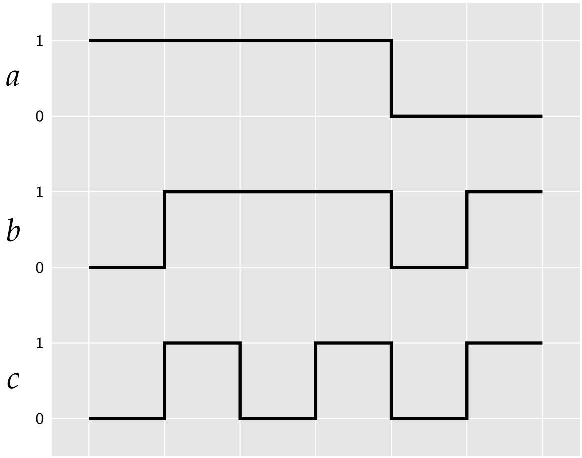
[1]LRE (EPITA), [2]SiPearl

CIAA — Akita — September 5, 2024

Context: Signals

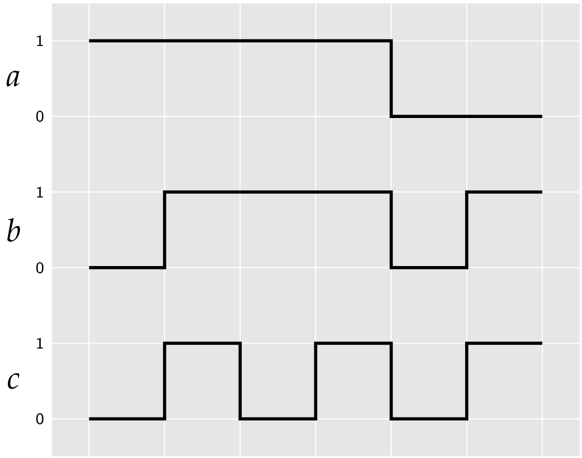


Context: Signals



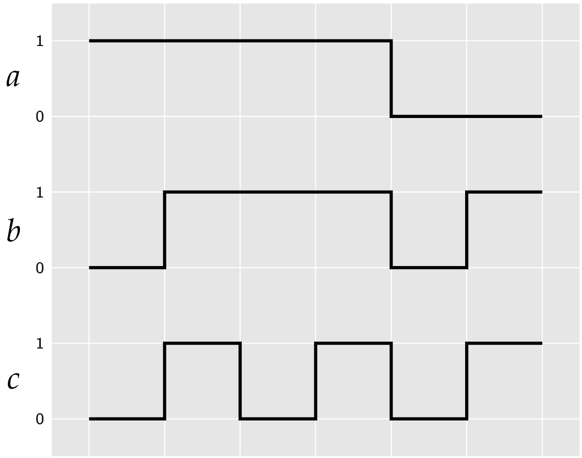
$$a\bar{b}\bar{c} \;;\; abc \;;\; ab\bar{c} \;;\; abc \;;\; \bar{a}\bar{b}\bar{c} \;;\; \bar{a}bc$$

Context: Signals



$a\bar{b}\bar{c}$; abc ; $ab\bar{c}$; abc ; $\bar{a}\bar{b}\bar{c}$; $\bar{a}bc$ } Word

Context: Signals

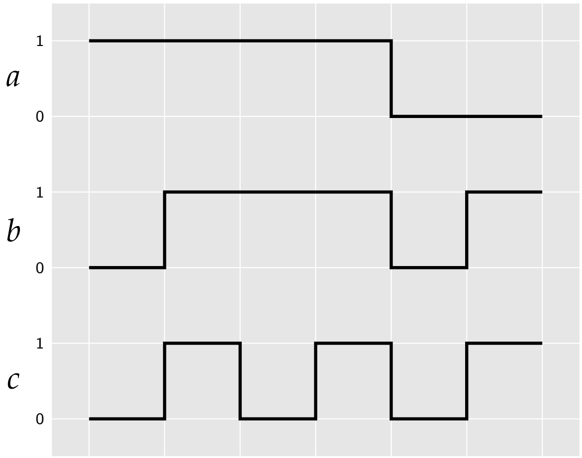


$a\bar{b}\bar{c}$; abc ; $ab\bar{c}$; abc ; $\bar{a}\bar{b}\bar{c}$; $\bar{a}bc$ } Word

Alphabet

$$\Sigma = 2^{\{\mathsf{a},\mathsf{b},\mathsf{c}\}}$$

Context: Signals



$a\bar{b}\bar{c}$; abc ; $ab\bar{c}$; abc ; $\bar{a}\bar{b}\bar{c}$; $\bar{a}bc$ } Word

Alphabet

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

Context: PSL[1] and SVA[2] translation to Büchi automata

$$\{(\mathtt{a} \wedge \mathtt{b})* : (\mathtt{c}\ ;\ !\,\mathtt{a})\}\,[\,]\!\!->\mathtt{G}\ \mathtt{c}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

{(a ∧ b)* : (c ; ! a)}[]->G c

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

```
{(a ∧ b)* : (c ; ! a)}[]-> G c
```

Linear-time Temporal Logic

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

{(a ∧ b)* : (c ; ! a)}[]->G c

Semi-Extended Regular Expressions

Linear-time Temporal Logic

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

$\{(\mathtt{a} \wedge \mathtt{b})\mathtt{*} : (\mathtt{c} ; \mathtt{!} \, \mathtt{a})\}\,\mathtt{[]\text{->}}\,\mathtt{G}\,\mathtt{c}$

Semi-Extended Regular Expressions

Linear-time Temporal Logic

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

$(a \wedge b)* : (c \; ; \; ! \, a)$

Semi-Extended Regular Expressions

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

$$(a \wedge b)* : (c \; ; \; ! \; a)$$

Semi-Extended Regular Expressions

- Regular operators: ";", "*", "|"

Alphabet is 2^{AP}

Boolean formulas as abbreviations

$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

Context: PSL[1] and SVA[2] translation to Büchi automata

$$(\texttt{a} \wedge \texttt{b})* : (\texttt{c} \; ; \; ! \, \texttt{a})$$

Semi-Extended Regular Expressions

- Regular operators: ";", "*", "|"
- Additional operators:
  - Intersection "&"
  - First Match "fm"
  - Fusion ":"

Alphabet is 2^{AP}

Boolean formulas as abbreviations

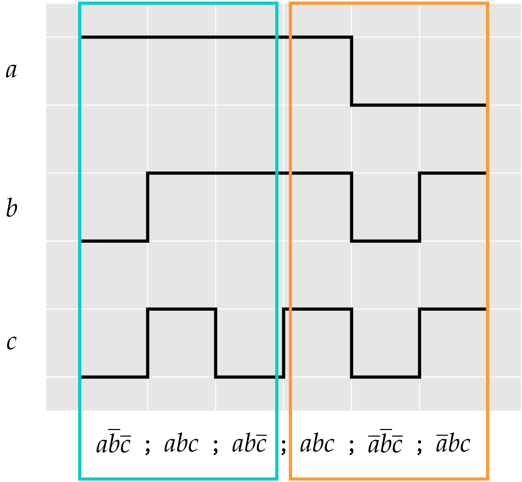
$$a \wedge b = \{abc, ab\bar{c}\}$$

[1] *Property Specification Language Reference Manual v1.1*, 2004.
[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, 2018.

SERE syntax: fusion operator

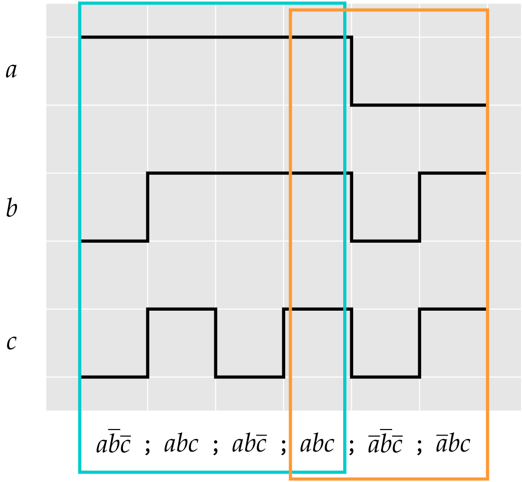
SERE syntax: fusion operator

Concatenation: $E \,;\, F$

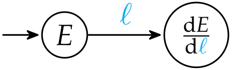


SERE syntax: fusion operator

Fusion: $E : F$



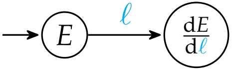
Regex to automaton translation

- Brzozowski, 1964 

$\ell \in \Sigma$

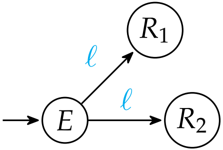
Regex to automaton translation

- Brzozowski, 1964



$$\frac{\mathrm{d}E}{\mathrm{d}\ell} = R_1 \vee R_2$$

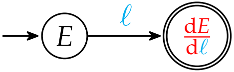
- Antimirov, 1996



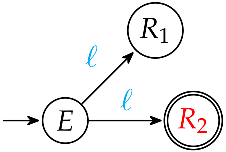
$$\frac{\partial E}{\partial \ell} = \{R_1, R_2\}$$

$\ell \in \Sigma$

Regex to automaton translation

- Brzozowski, 1964 

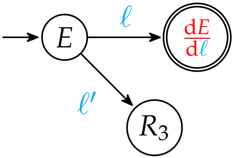
$\frac{dE}{d\ell} = R_1 \vee R_2$

- Antimirov, 1996 

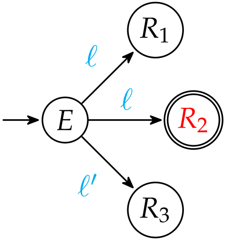
$\frac{\partial E}{\partial \ell} = \{R_1, R_2\}$, $\varepsilon \models R_2$

$\ell \in \Sigma$

Regex to automaton translation

- Brzozowski, 1964 

$$\frac{dE}{d\ell} = R_1 \vee R_2$$

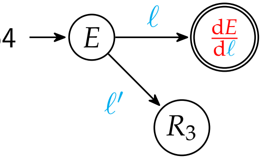
- Antimirov, 1996 

$$\frac{\partial E}{\partial \ell} = \{R_1, R_2\}\ ,\ \varepsilon \models R_2$$

$\ell \in \Sigma$

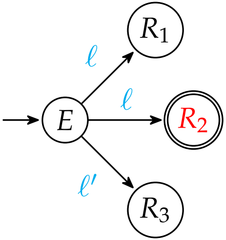
Regex to automaton translation

- Brzozowski, 1964



$$\frac{\mathrm{d}E}{\mathrm{d}\ell} = R_1 \vee R_2$$

- Antimirov, 1996



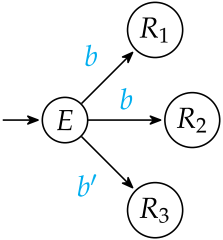
$$\frac{\partial E}{\partial \ell} = \{R_1, R_2\} \;,\; \varepsilon \models R_2$$

$\ell \in \Sigma$

$$\mathsf{LF}(E) = \{(\ell, R_1), (\ell, R_2), (\ell', R_3)\}$$

Regex to automaton translation

- Antimirov, 1996



$$\mathsf{LF}(E) = \{(b, R_1), (b, R_2), (b', R_3)\}$$

$b \subseteq 2^{AP}$

Antimirov's Linear Forms

$$\begin{aligned}
\mathsf{LF}(\bot) &= \varnothing \\
\mathsf{LF}(\varepsilon) &= \varnothing \\
\mathsf{LF}(\ell) &= \{(\ell, \varepsilon)\} \\
\mathsf{LF}(r_1 \vee r_2) &= \mathsf{LF}(r_1) \cup \mathsf{LF}(r_2) \\
\mathsf{LF}(r^\star) &= \mathsf{LF}(r) \,;\, r^\star \\
\mathsf{LF}(r_1 \,;\, r_2) &= (\mathsf{LF}(r_1) \,;\, r_2) \cup (\lambda(r_1) \,;\, \mathsf{LF}(r_2))
\end{aligned}$$

Antimirov's Linear Forms

$$
\begin{aligned}
\mathsf{LF}(\bot) &= \varnothing \\
\mathsf{LF}(\varepsilon) &= \varnothing \\
\mathsf{LF}(\ell) &= \{(\ell, \varepsilon)\} \\
\mathsf{LF}(r_1 \lor r_2) &= \mathsf{LF}(r_1) \cup \mathsf{LF}(r_2) \\
\mathsf{LF}(r^\star) &= \mathsf{LF}(r) \,;\, r^\star \\
\mathsf{LF}(r_1 \,;\, r_2) &= (\mathsf{LF}(r_1) \,;\, r_2) \cup (\lambda(r_1) \,;\, \mathsf{LF}(r_2))
\end{aligned}
$$

Antimirov's Linear Forms

$$
\begin{aligned}
\mathsf{LF}(\bot) &= \varnothing \\
\mathsf{LF}(\varepsilon) &= \varnothing \\
\mathsf{LF}(b) &= \{(b, \varepsilon)\} \\
\mathsf{LF}(r_1 \vee r_2) &= \mathsf{LF}(r_1) \cup \mathsf{LF}(r_2) \\
\mathsf{LF}(r^\star) &= \mathsf{LF}(r) \,;\, r^\star \\
\mathsf{LF}(r_1 \,;\, r_2) &= (\mathsf{LF}(r_1) \,;\, r_2) \cup (\lambda(r_1) \,;\, \mathsf{LF}(r_2))
\end{aligned}
$$

Antimirov's Linear Forms

$$
\begin{aligned}
\mathsf{LF}(\bot) &= \varnothing \\
\mathsf{LF}(\varepsilon) &= \varnothing \\
\mathsf{LF}(b) &= \{(b, \varepsilon)\} \\
\mathsf{LF}(r_1 \vee r_2) &= \mathsf{LF}(r_1) \cup \mathsf{LF}(r_2) \\
\mathsf{LF}(r^\star) &= \mathsf{LF}(r) \mathbin{;} r^\star \\
\mathsf{LF}(r_1 \mathbin{;} r_2) &= (\mathsf{LF}(r_1) \mathbin{;} r_2) \cup (\lambda(r_1) \mathbin{;} \mathsf{LF}(r_2)) \\
\mathsf{LF}(r_1 : r_2) &= (\mathsf{LF}(r_1) : r_2) \cup \left\{ (p_i \wedge p_j, s_j) \;\middle|\; \begin{array}{l} (p_i, s_i) \in \mathsf{LF}(r_1),\ \lambda(s_i) = \varepsilon, \\ (p_j, s_j) \in \mathsf{LF}(r_2) \end{array} \right\} \\
\mathsf{LF}(r_1 \wedge r_2) &= \{(p_i \wedge p_j, s_i \wedge s_j) \mid (p_i, s_i) \in \mathsf{LF}(r_1),\ (p_j, s_j) \in \mathsf{LF}(r_2)\} \\
\mathsf{LF}(\mathsf{fm}(r)) &= \{(p_i, \mathsf{fm}(s_i)) \mid (p_i, s_i) \in \mathrm{det}(\mathsf{LF}(r))\}
\end{aligned}
$$

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

```
Q, δ, F ← {φ}, ∅, ∅;
todo.push(φ);
while todo ≠ ∅ do
    f ← todo.pop();
    foreach (p, s) ∈ LF (f) do
        if s ∉ Q then
            Q ← Q ∪ {s};
            todo.push (s);
            if ε ⊨ s then
                F ← F ∪ {s};
        δ ← δ ∪ {f -p-> s};
return ⟨Q, δ, φ, F⟩;
```

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

$Q, \delta, F \leftarrow \{\phi\}, \varnothing, \varnothing$;
todo.push(ϕ);
while todo $\neq \varnothing$ **do**
  $f \leftarrow$ todo.pop();
  foreach $(p, s) \in$ LF (f) **do**
    if $s \notin Q$ **then**
      $Q \leftarrow Q \cup \{s\}$;
      todo.push (s);
      if $\varepsilon \models s$ **then**
        $F \leftarrow F \cup \{s\}$;
    $\delta \leftarrow \delta \cup \{f \xrightarrow{p} s\}$;
return $\langle Q, \delta, \phi, F \rangle$;

Initial state is the input formula ϕ

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

```
Q, δ, F ← {φ}, ∅, ∅;
todo.push(φ);
while todo ≠ ∅ do
    f ← todo.pop();
    foreach (p, s) ∈ LF (f) do
        if s ∉ Q then
            Q ← Q ∪ {s};
            todo.push (s);
            if ε ⊨ s then
                F ← F ∪ {s};
        δ ← δ ∪ {f -p-> s};
return ⟨Q, δ, φ, F⟩;
```

Initial state is the input formula ϕ

Loop over LF pairs

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

```
Q, δ, F ← {φ}, ∅, ∅;
todo.push(φ);
while todo ≠ ∅ do
    f ← todo.pop();
    foreach (p, s) ∈ LF(f) do
        if s ∉ Q then
            Q ← Q ∪ {s};
            todo.push(s);
            if ε ⊨ s then
                F ← F ∪ {s};
        δ ← δ ∪ {f --p--> s};
return ⟨Q, δ, φ, F⟩;
```

Initial state is the input formula ϕ

Loop over LF pairs

States are labeled by SEREs

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

$Q, \delta, F \leftarrow \{\phi\}, \varnothing, \varnothing$;
todo.push(ϕ);
while todo $\neq \varnothing$ **do**
  $f \leftarrow$ todo.pop();
  foreach $(p, s) \in$ LF (f) **do**
    if $s \notin Q$ **then**
      $Q \leftarrow Q \cup \{s\}$;
      todo.push (s);
      if $\varepsilon \models s$ **then**
        $F \leftarrow F \cup \{s\}$;
    $\delta \leftarrow \delta \cup \{f \xrightarrow{p} s\}$;
return $\langle Q, \delta, \phi, F \rangle$;

Initial state is the input formula ϕ

Loop over LF pairs

States are labeled by SEREs

Identify accepting states

Algorithm 1: Translation

input : A SERE ϕ
output: An automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\phi)$

$Q, \delta, F \leftarrow \{\phi\}, \varnothing, \varnothing$;
todo.push(ϕ);
while todo $\neq \varnothing$ **do**
    $f \leftarrow$ todo.pop();
    foreach $(p, s) \in$ LF (f) **do**
        if $s \notin Q$ **then**
            $Q \leftarrow Q \cup \{s\}$;
            todo.push (s);
            if $\varepsilon \models s$ **then**
                $F \leftarrow F \cup \{s\}$;
        $\delta \leftarrow \delta \cup \{f \xrightarrow{p} s\}$;
return $\langle Q, \delta, \phi, F \rangle$;

Initial state is the input formula ϕ

Loop over LF pairs

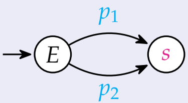
States are labeled by SEREs

Identify accepting states

Two simplifications on linear forms

Unique Suffix

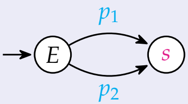
$$\mathsf{LF}(E) = \{(p_1, s), (p_2, s)\}$$



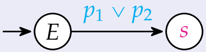
Two simplifications on linear forms

Unique Suffix

$$\mathsf{LF}(E) = \{(p_1, s), (p_2, s)\}$$



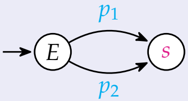
$$\mathsf{US}(\mathsf{LF}(E)) = \{(p_1 \vee p_2, s)\}$$



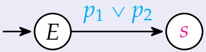
Two simplifications on linear forms

Unique Suffix

$$\mathsf{LF}(E) = \{(p_1, s), (p_2, s)\}$$

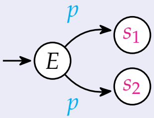


$$\mathsf{US}(\mathsf{LF}(E)) = \{(p_1 \vee p_2, s)\}$$



Unique Prefix

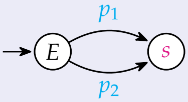
$$\mathsf{LF}(E) = \{(p, s_1), (p, s_2)\}$$



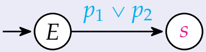
Two simplifications on linear forms

Unique Suffix

$$\mathsf{LF}(E) = \{(p_1, s), (p_2, s)\}$$

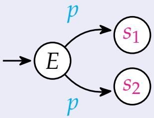


$$\mathsf{US}(\mathsf{LF}(E)) = \{(p_1 \vee p_2, s)\}$$

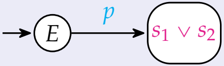


Unique Prefix

$$\mathsf{LF}(E) = \{(p, s_1), (p, s_2)\}$$



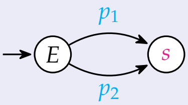
$$\mathsf{UP}(\mathsf{LF}(E)) = \{(p, s_1 \vee s_2)\}$$



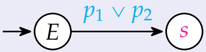
Two simplifications on linear forms

Unique Suffix

$$\mathsf{LF}(E) = \{(p_1, s), (p_2, s)\}$$

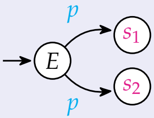


$$\mathsf{US}(\mathsf{LF}(E)) = \{(p_1 \vee p_2, s)\}$$

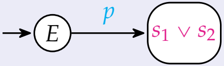


Unique Prefix

$$\mathsf{LF}(E) = \{(p, s_1), (p, s_2)\}$$



$$\mathsf{UP}(\mathsf{LF}(E)) = \{(p, s_1 \vee s_2)\}$$



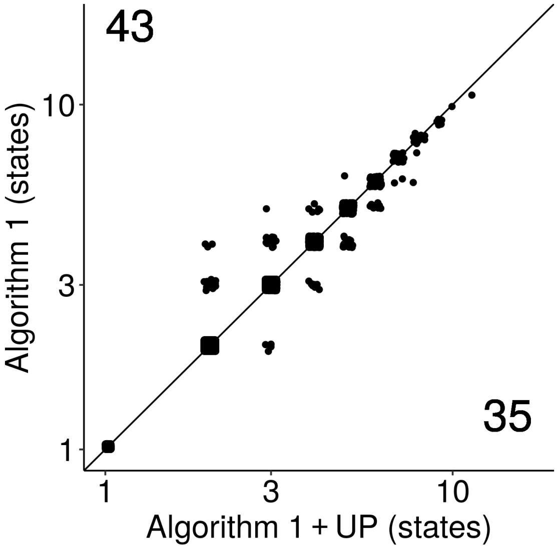
Note that this is not equivalent to a determinization in our case

Benchmarking UP effects

Benchmark dataset

12500 SEREs, randomly generated with Spot [a]. We enforced a certain heterogeneity in the dataset.

[a] https://spot.lre.epita.fr



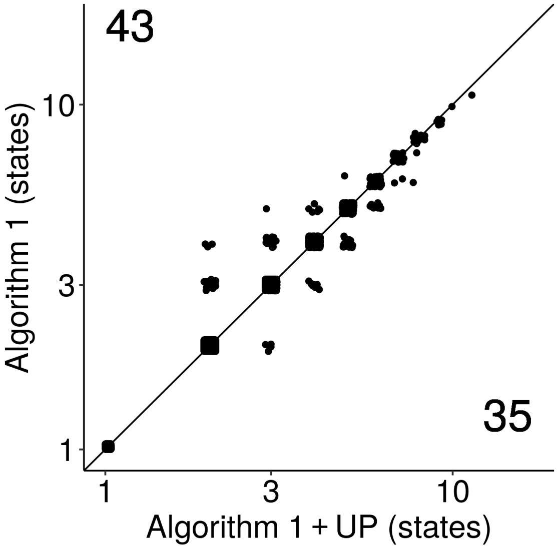
Benchmarking UP effects

Benchmark dataset

12500 SEREs, randomly generated with Spot [a]. We enforced a certain heterogeneity in the dataset.

[a] https://spot.lre.epita.fr

Some jitter was added to the plot



An interesting Linear Form property

Two expressions can have the same linear form:

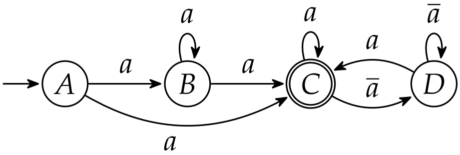
Example

$$\mathsf{LF}(\mathsf{a}*) = \mathsf{LF}(\mathsf{a} \ ; \ \mathsf{a}*) = \{(\mathsf{a}, \mathsf{a}*)\}$$

Property (Antimirov, 1996)

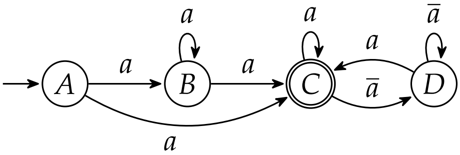
$$\mathsf{LF}(E) = \mathsf{LF}(F) \implies \mathscr{L}(E) \backslash \{\varepsilon\} = \mathscr{L}(F) \backslash \{\varepsilon\}$$

Improving the translation

Algo 1: 

Improving the translation

Algo 1:

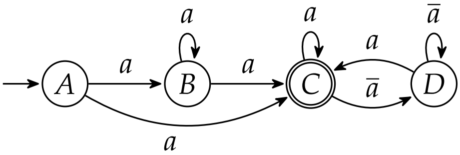


Notice

$$\mathsf{LF}(A) = \mathsf{LF}(B)$$

$$\mathsf{LF}(C) = \mathsf{LF}(D)$$

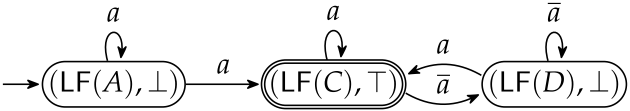
Improving the translation

Algo 1: 

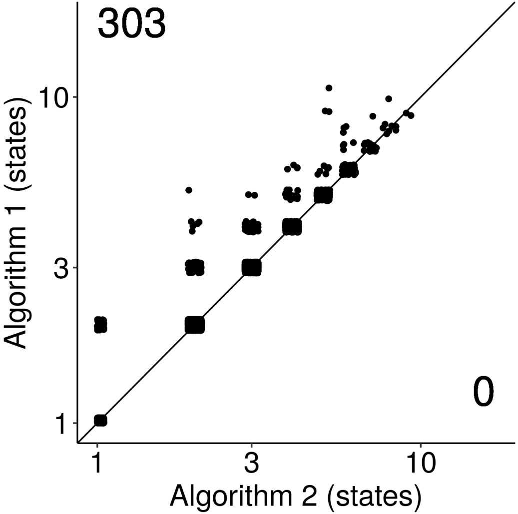
Notice

$$\mathsf{LF}(A) = \mathsf{LF}(B)$$

$$\mathsf{LF}(C) = \mathsf{LF}(D)$$

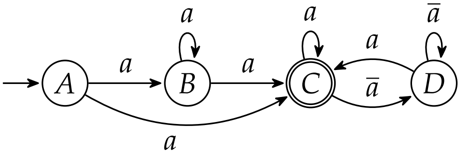
Algo 2: labeling with linear forms 

Algorithm 2 is an improvement



Improving the translation

Algo 1:

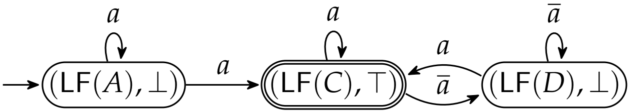


Notice

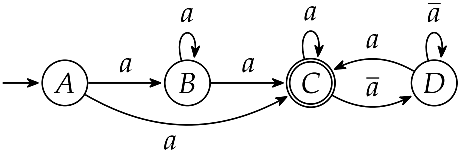
$$\mathsf{LF}(A) = \mathsf{LF}(B)$$

$$\mathsf{LF}(C) = \mathsf{LF}(D)$$

Algo 2: labeling with linear forms



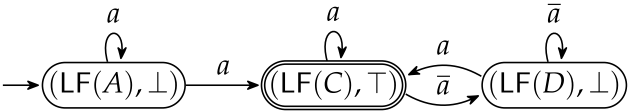
Improving the translation

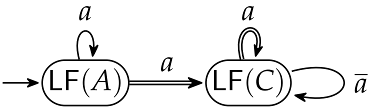
Algo 1: 

Notice

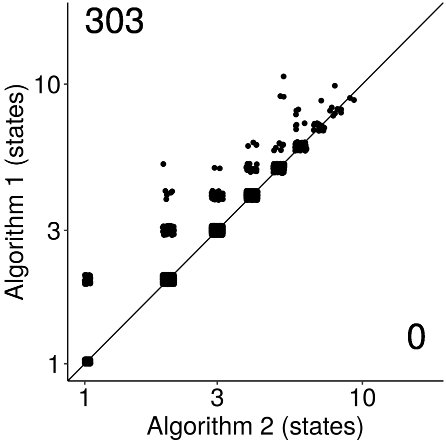
$$\mathsf{LF}(A) = \mathsf{LF}(B)$$

$$\mathsf{LF}(C) = \mathsf{LF}(D)$$

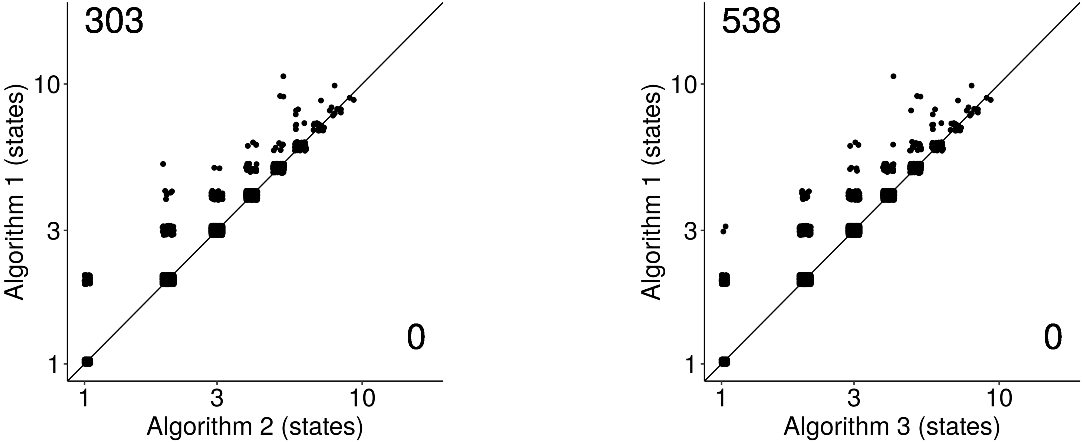
Algo 2: labeling with linear forms 

Algo 3: using transition-based acceptance 

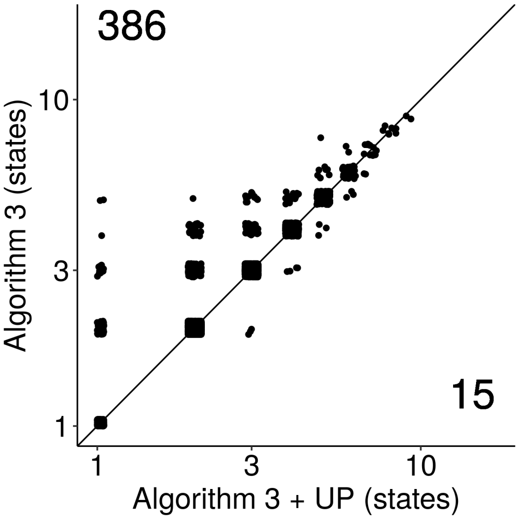
Algorithm 3 is better



Algorithm 3 is better



Effect of UP on transition-based acceptance



Wrapping up

Contributions

- Extended Antimirov's linear forms
  - Support alphabet $\Sigma = 2^{AP}$
  - Support SERE operators
- Evaluation of UP / US simplifications' performance
- Labeling of automaton states with linear forms
- Translation using transition-based acceptance

Wrapping up

Contributions

- Extended Antimirov's linear forms
  - Support alphabet $\Sigma = 2^{AP}$
  - Support SERE operators
- Evaluation of UP / US simplifications' performance
- Labeling of automaton states with linear forms
- Translation using transition-based acceptance

Future work

Build on this work to translate PSL / SVA to Büchi automata, and evaluate performance of translation in a model checking context.