Solving String Constraints: From the Beginning and the End

Philipp Rümmer, Uppsala University

Simons Institute, Spring 2021 Workshops
March 24th, 2021, AoE

Joint work with ...

- Parosh Aziz Abdulla
- Moh. Faouzi Atig
- Yu-Fang Chen
- Taolue Chen
- Bui Phi Diep
- Matthew Hague
- Jinlong He
- Hossein Hojjat
- Lukás Holík
- Denghang Hu
- Petr Janků
- Anthony W. Lin
- Ahmed Rezine
- Ali Shamakhi
- Jari Stenman
- Albin Stjerna
- Tomás Vojnar
- Zhilin Wu
- *... and others*

Plan

- String constraints, applications
- Algorithms/tools for SAT modulo strings:
  - Norn
  - OSTRICH

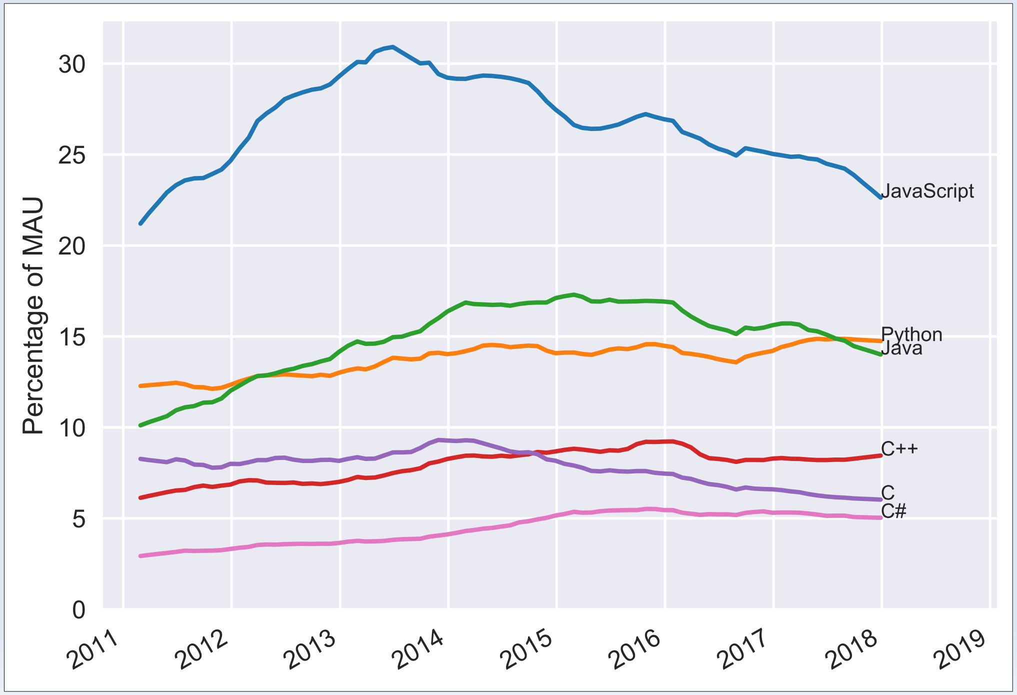

Starting from the Beginning
…

Strings: Datatype #1

```
object HelloWorld extends App {
  println("Hello Simons Institute!")
}
```

Languages

- Modern languages (e.g., JavaScript, PHP, Python) have rich libraries for strings
  - Do more with fewer LOC
  - Strings used everywhere



https://www.benfrederickson.com/ranking-programming-languages-by-github-users/

Common Operations

- Concatenation, splitting
- Length
- Regular expressions
- Search/replace
- Conversions: strings ↔ numbers
- Encoding/decoding, escaping
- *etc.*

Common Operations

- Concatenation, splitting
- Length
- Regular expressions
- Search/replace
- Conversions: strings ↔ numbers
- Encoding/decoding, escaping
- *etc.*

Tricky features:
eager/lazy matching, back-references, anchors, etc.

Security-Critical Operations

- Input sanitisation, validation
- Query generation for databases
- Data handling: XML, JSON, HTML
- Dynamic code generation
- Dynamic class loading, method invocation

Strings = Data + Code

- Input sanitisation, validation
- Query generation for databases
- Data handling: XML, JSON, HTML
- Dynamic code generation
- Dynamic class loading, method invocation

A Subtle XSS Vulnerability

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

A Subtle XSS Vulnerability

Input string:
cat name

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

A Subtle XSS Vulnerability

HTML escape:
& → &
' → '
etc.

Input string:
cat name

JavaScript escape:
' → \'

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

A Subtle XSS Vulnerability

HTML escape:
& → &
' → '
etc.

Input string:
cat name

JavaScript escape:
' → \'

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

Implicit HTML **un**escape
of the `onclick`
attribute:
& → &

An XSS Vulnerability (2)

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

One possible attack

Choose `cat` to be `');alert(1);//`

Generated HTML string is then:

```
<button onclick="createCatList('&#39;);alert(1);//')">
&#39;);alert(1);//</button>
```

An XSS Vulnerability (2)

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

One possible attack

Choose `cat` to be `');`

This will be **unescaped** to
`createCatList('');alert(1);//')`

Generated HTML string is then:

```
<button onclick="createCatList('&#39;);alert(1);//')">
&#39;);alert(1);//</button>
```

An XSS Vulnerability (2)

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="creat
  y + '\')">' + x + '</bu
```

Vulnerability since escape functions are applied in **wrong order**

One possible atta

Choose `cat` to be `');`

This will be **unescaped** to `createCatList('');alert(1);//')`

Generated HTML string is then:

```
<button onclick="createCatList('&#39;);alert(1);//')">
&#39;);alert(1);//</button>
```

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$x = htmlEscape(cat)$$

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$x = htmlEscape(cat)$$
$$\wedge\, y = escape(x)$$

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$x = htmlEscape(cat)$$
$$\wedge\, y = escape(x)$$
$$\wedge\, z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$x = htmlEscape(cat)$$
$$\wedge\, y = escape(x)$$
$$\wedge\, z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$
$$\wedge\, innerHTML = htmlUnescape(z)$$

String Formula Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$
\begin{aligned}
& x = htmlEscape(cat) \\
\wedge\ & y = escape(x) \\
\wedge\ & z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3 \\
\wedge\ & innerHTML = htmlUnescape(z) \\
\wedge\ & attack(innerHTML)
\end{aligned}
$$

Wishlist: Solvers with ...

- Concatenation, splitting
- Length
- Regular expressions
- Search/replace
- Conversions: strings ↔ numbers
- Encoding/decoding, escaping

- ASCII (7/8 bit), Unicode (21 bit)
- Representations: UTF-8, UTF-16, UTF-32

Challenges

- **Q1**: What is decidable about strings?
- **Q2**: What is the right string logic/fragment?
- **Q3**: How to build efficient solvers?

```
(theory Strings

:smt-lib-version 2.6
:written_by "Cesare Tinelli, Clark Barrett, and Pascal Fontaine"
:date "2020-02-11"

:notes
"This is a theory of character strings and regular expressions over an alphabet
 consisting of Unicode characters. It is not meant to be used in isolation but
 in combination with Ints, the theory of integer numbers.
"

:notes
"The theory is based on an initial proposal by Nikolaj Bjørner, Vijay Ganesh,
 Raphaël Michel and Margus Veanes at SMT 2012.
 The following people, in alphabetical order, have contributed further suggestions
 that helped shape the current version of the theory (with our apologies for any,
 unintentional, omissions):
 Kshitij Bansal, Murphy Berzish, Nikolaj Bjørner, David Cok, Levent Erkok,
 Andrew Gacek, Vijay Ganesh, Alberto Griggio, Joxan Jaffar, Anthony Lin, Andres
 Nötzli, Andrew Reynolds, Philipp Rümmer, Margus Veanes, and Tjark Weber.
"
```

Some String Solvers

- Hampi
- Kaluza
- Stranger
- Gecode+S
- GStrings
- Z3
- Z3-str/2/3/4
- CVC4
- Norn
- S3/p/#
- TRAU
- Sloth
- OSTRICH
- ABC
- Woorpje
- BEK, REX
- SLOG, SLENT
- *to be continued ...*


SMS
(Satisfiability Modulo Strings)

Solving String Constraints?

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

Equations, Concatenation

Solving String Constraints?

This is already in **solved form**

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

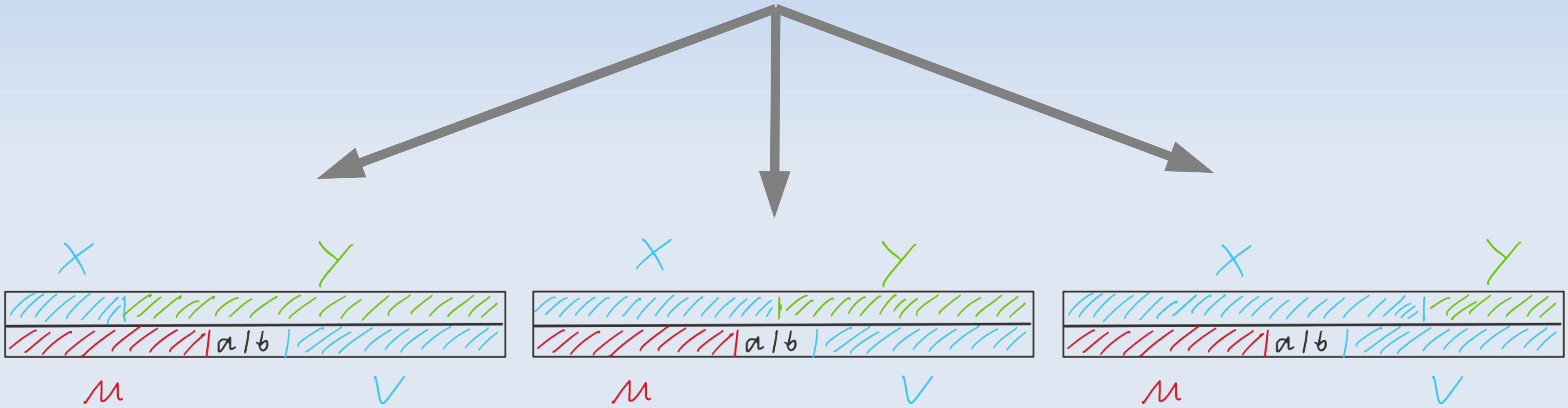
Equations, Concatenation

Solutions of this Equation?

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$

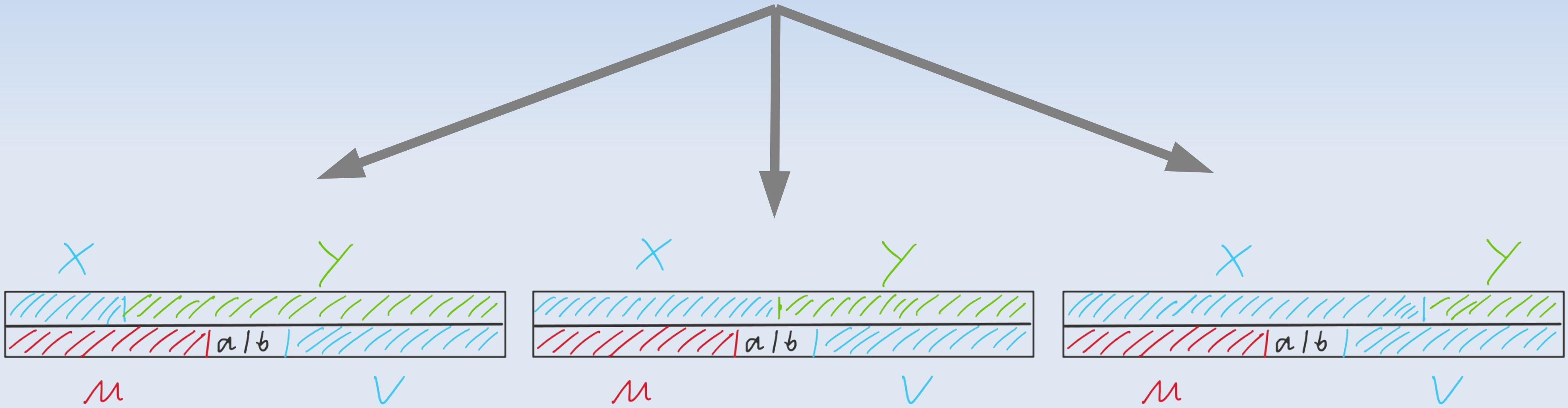
Solutions of this Equation?

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$



Solutions of this Equation?

$$x \cdot y = u \cdot \text{‘}ab\text{’} \cdot v$$



$$x = u_1$$
$$\wedge\ y = u_2 \cdot \text{‘}ab\text{’} \cdot v$$
$$\wedge\ u = u_1 \cdot u_2$$

$$x = u \cdot \text{‘}a\text{’}$$
$$\wedge\ y = \text{‘}b\text{’} \cdot v$$

$$x = u \cdot \text{‘}ab\text{’} \cdot v_1$$
$$\wedge\ y = v_2$$
$$\wedge\ v = v_1 \cdot v_2$$

Nielsen's transformation

(aka Levi's lemma)

Theorem

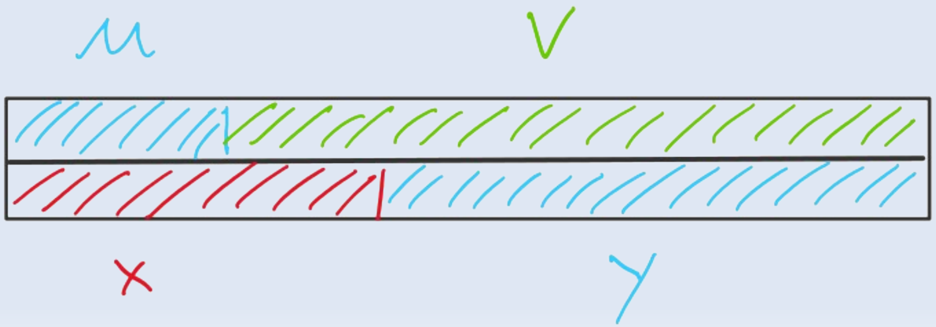
$$uv = xy \quad \Leftrightarrow \quad \begin{cases} \exists t.\ \ u = xt \land y = tv;\ \text{or} \\ \exists t.\ \ x = ut \land v = ty \end{cases}$$

Nielsen's transformation

(aka Levi's lemma)

Theorem

$$uv = xy \quad \Leftrightarrow \quad \begin{cases} \exists t. \;\; u = xt \wedge y = tv; \;\; \text{or} \\ \exists t. \;\; x = ut \wedge v = ty \end{cases}$$

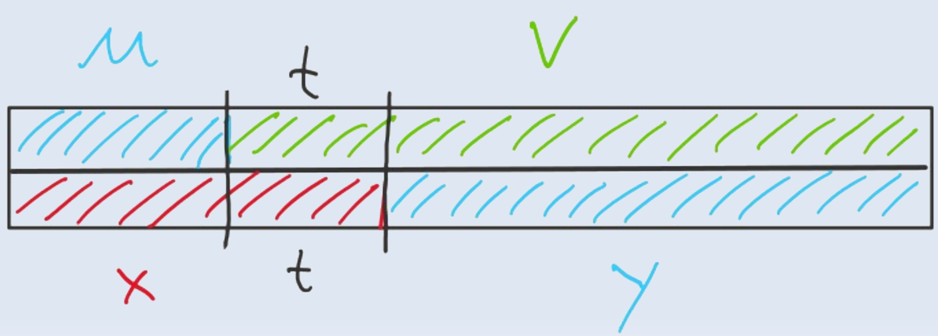


Nielsen's transformation

(aka Levi's lemma)

Theorem

$$uv = xy \quad \Leftrightarrow \quad \begin{cases} \exists t.\ \ u = xt \wedge y = tv; \text{ or} \\ \exists t.\ \ x = ut \wedge v = ty \end{cases}$$



How about this one?

$$x \cdot y = y \cdot z$$

How about this one?

$$x \cdot y = y \cdot z$$

$$x = y \cdot t \qquad y = x \cdot t$$

$$z = t \cdot y \qquad y = t \cdot z$$

How about this one?

$$x \cdot y = y \cdot z$$

$$x = y \cdot t$$
$$z = t \cdot y$$

$$\left.\begin{aligned} y &= x \cdot t \\ y &= t \cdot z \end{aligned}\right\} \quad x \cdot t = t \cdot z$$

How about this one?

$$x \cdot y = y \cdot z$$

$$x = y \cdot t$$
$$z = t \cdot y$$

$$\left.\begin{aligned} y &= x \cdot t \\ y &= t \cdot z \end{aligned}\right\} \quad x \cdot t = t \cdot z$$

Cycle!

What can be done?

Ignore cycles and hope for the best!

Identify fragments for which NT is guaranteed to terminate

- Acyclic; straight-line; chain-free; *etc.*

Develop actual decision procedures ...

- Makanin's method
- Recompression

What can be done?

Ignore cycles and hope for the best!

Identify fragments for which NT is guaranteed to terminate

- Acyclic; straight-line; chain-free; *etc.*

Develop actual decision procedures ...

- Makanin's method
- Recompression

Not implementable (?)

What can be done?

Ignore cycles and hope for the best!

Identify fragments for which NT is guaranteed to terminate

- Acyclic; straight-line; chain-free; *etc.*

Develop actual decision procedures ...

- Makanin's method
- Recompression

Not implementable (?)

Some String SMT Solvers

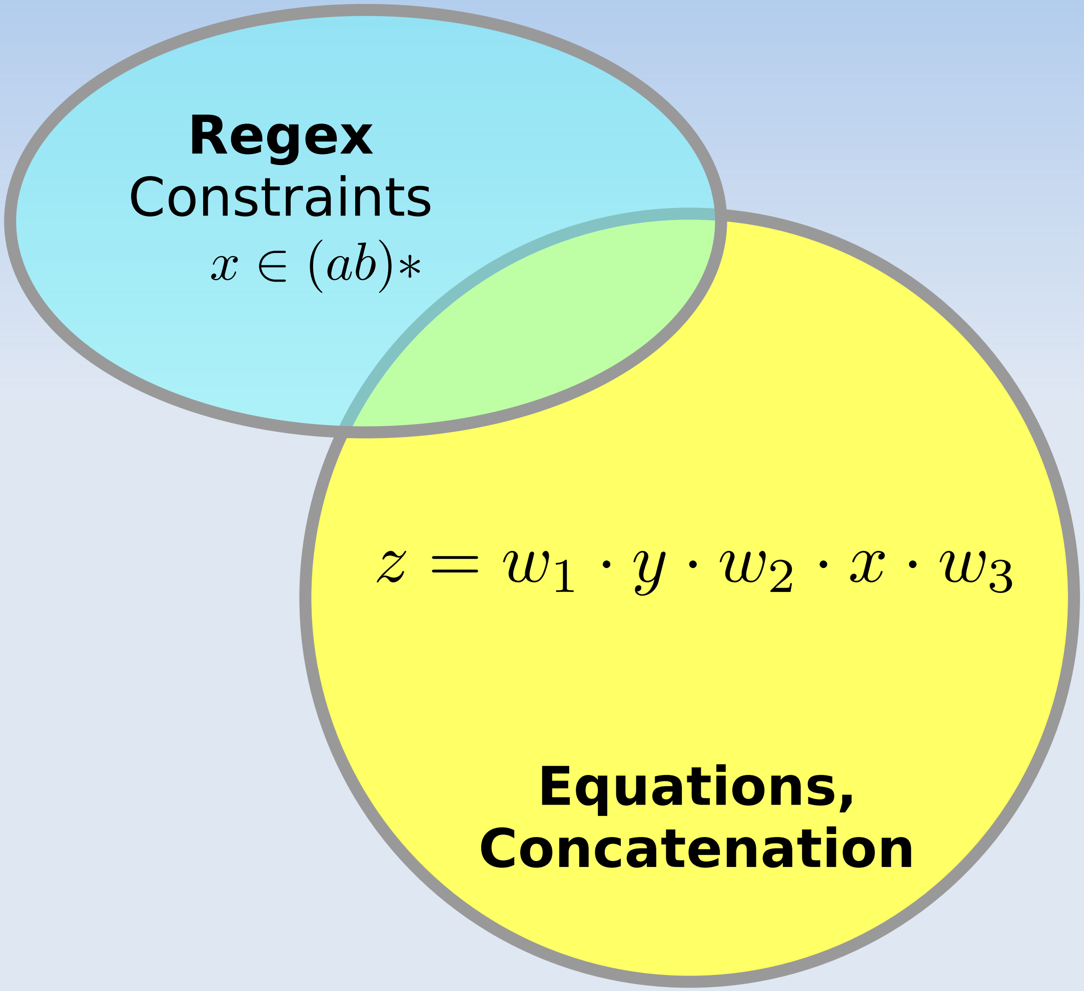
- Hampi
- Kaluza
- Stranger
- Gecode+S
- GStrings
- Z3
- Z3-str/2/3/4
- CVC4
- Norn
- S3/p/#
- TRAU
- Sloth
- OSTRICH
- ABC
- Woorpje
- BEK, REX
- SLOG, SLENT
- *to be continued ...*

Combinations ...

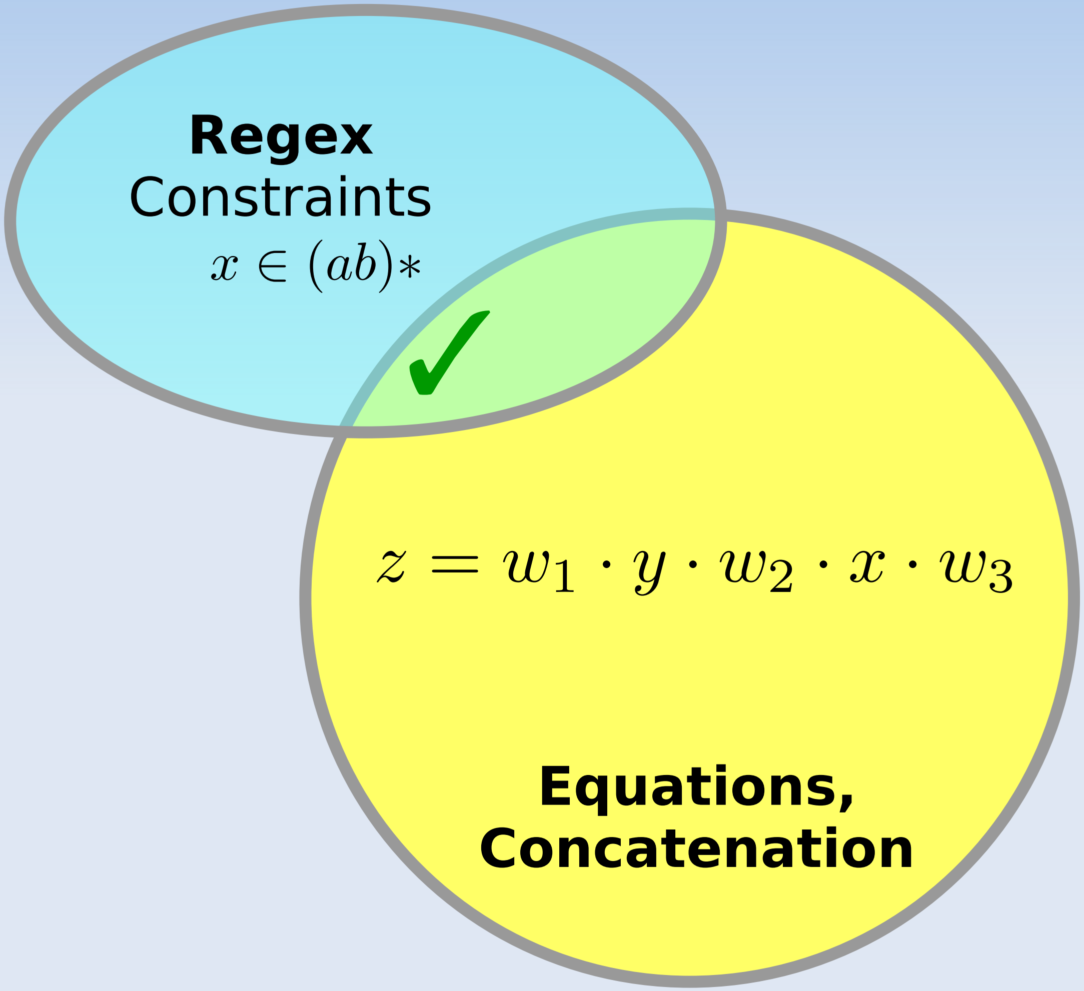
$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

Equations, Concatenation

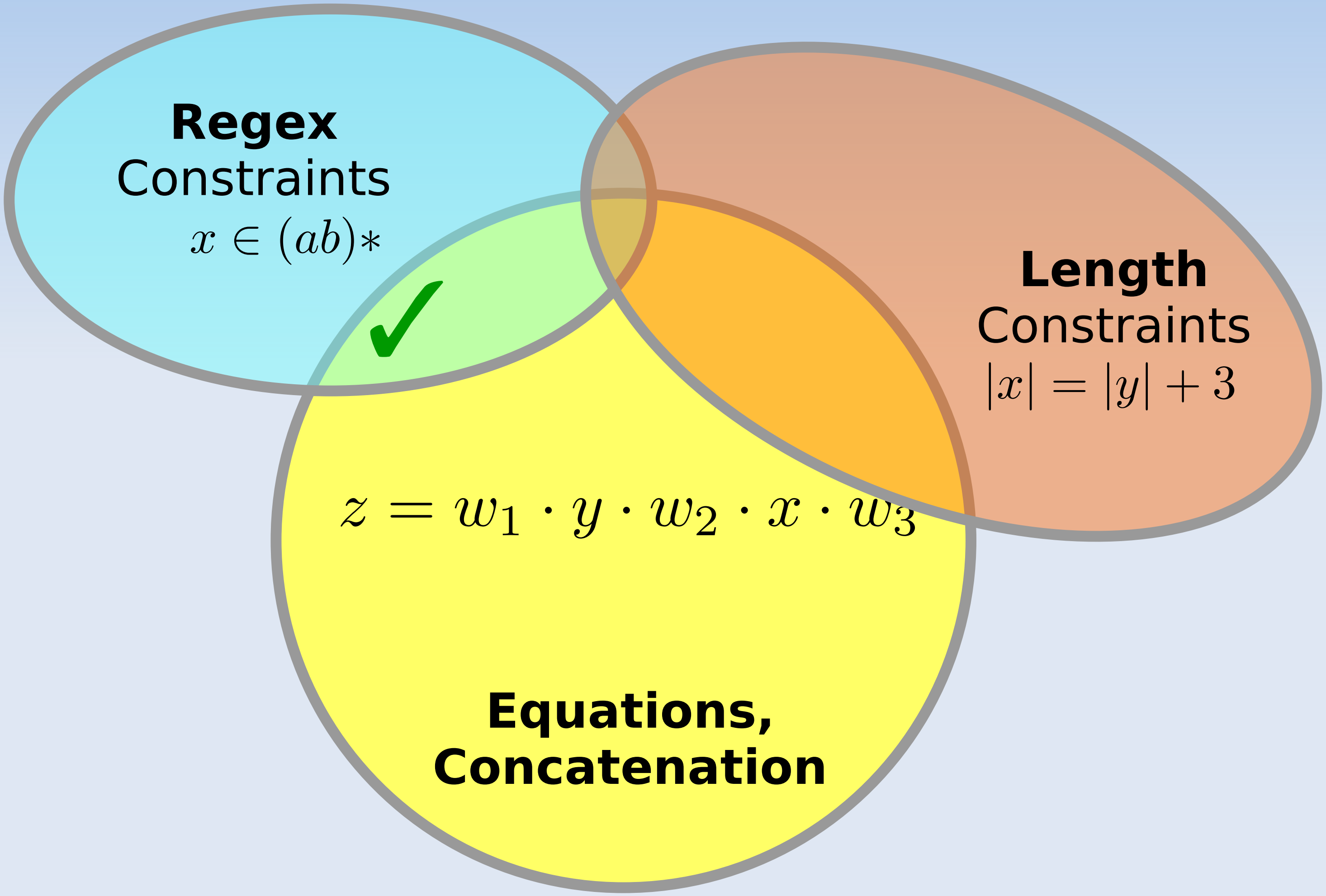
Combinations …



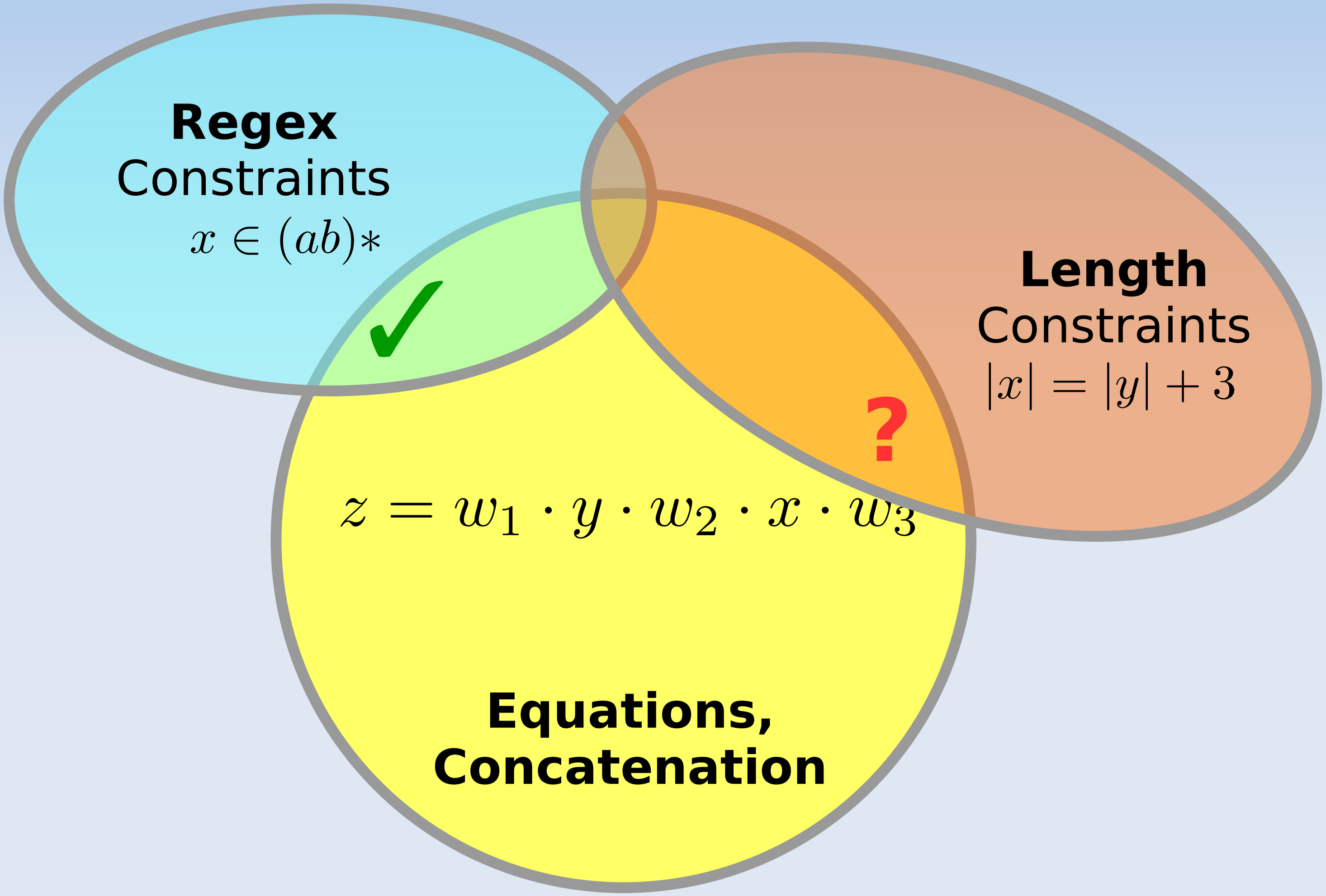
Combinations ...



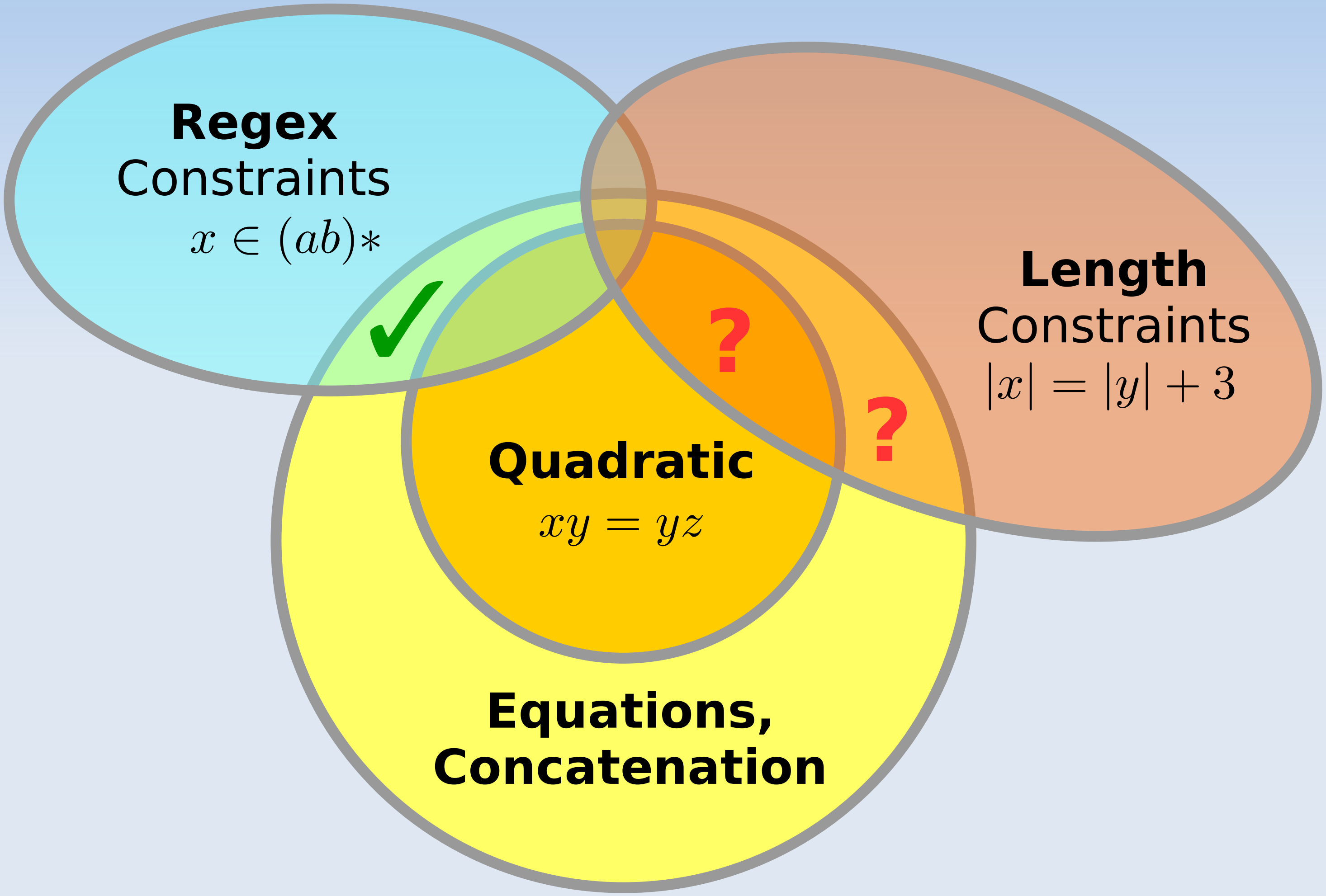
Combinations ...



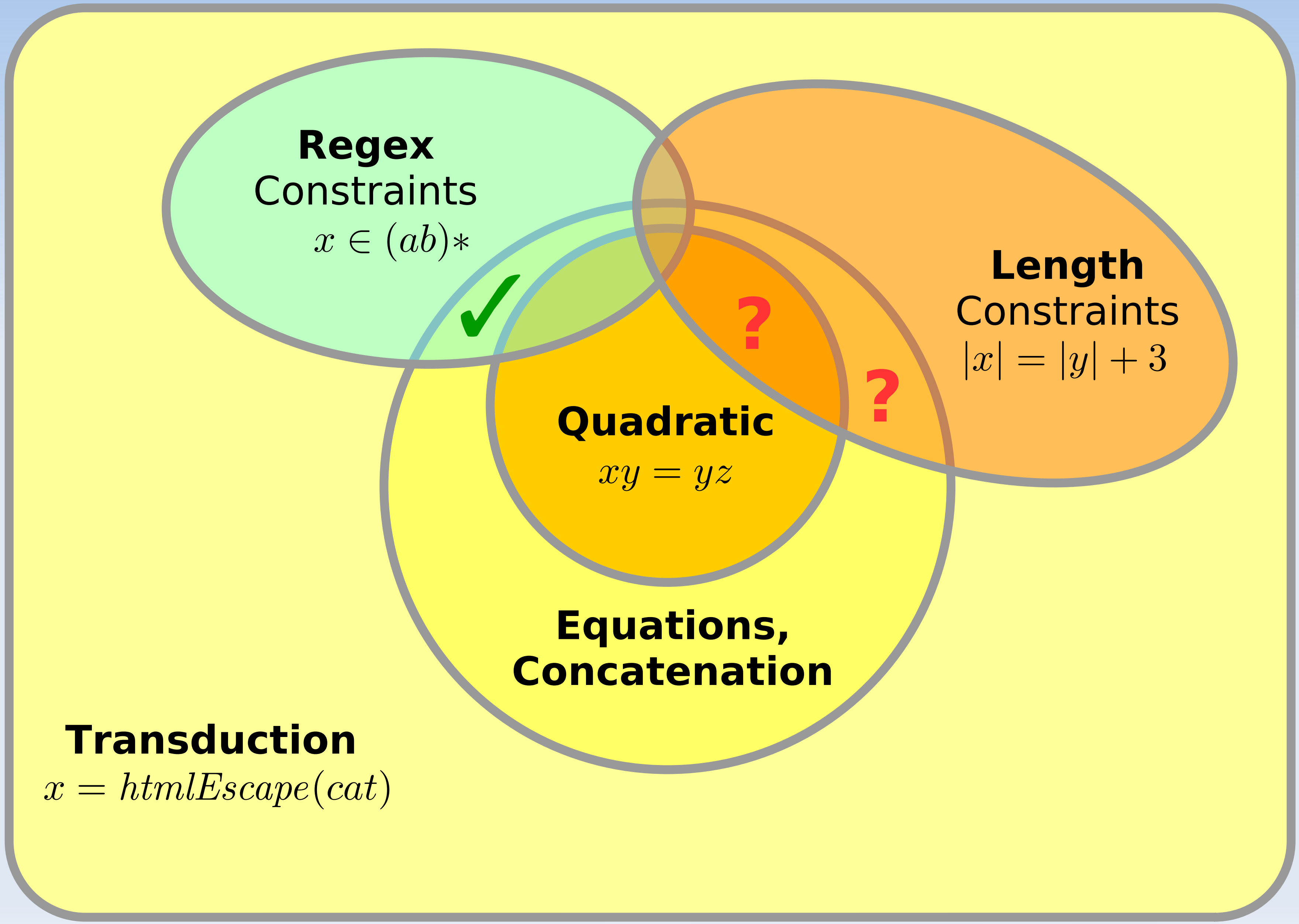
Combinations ...



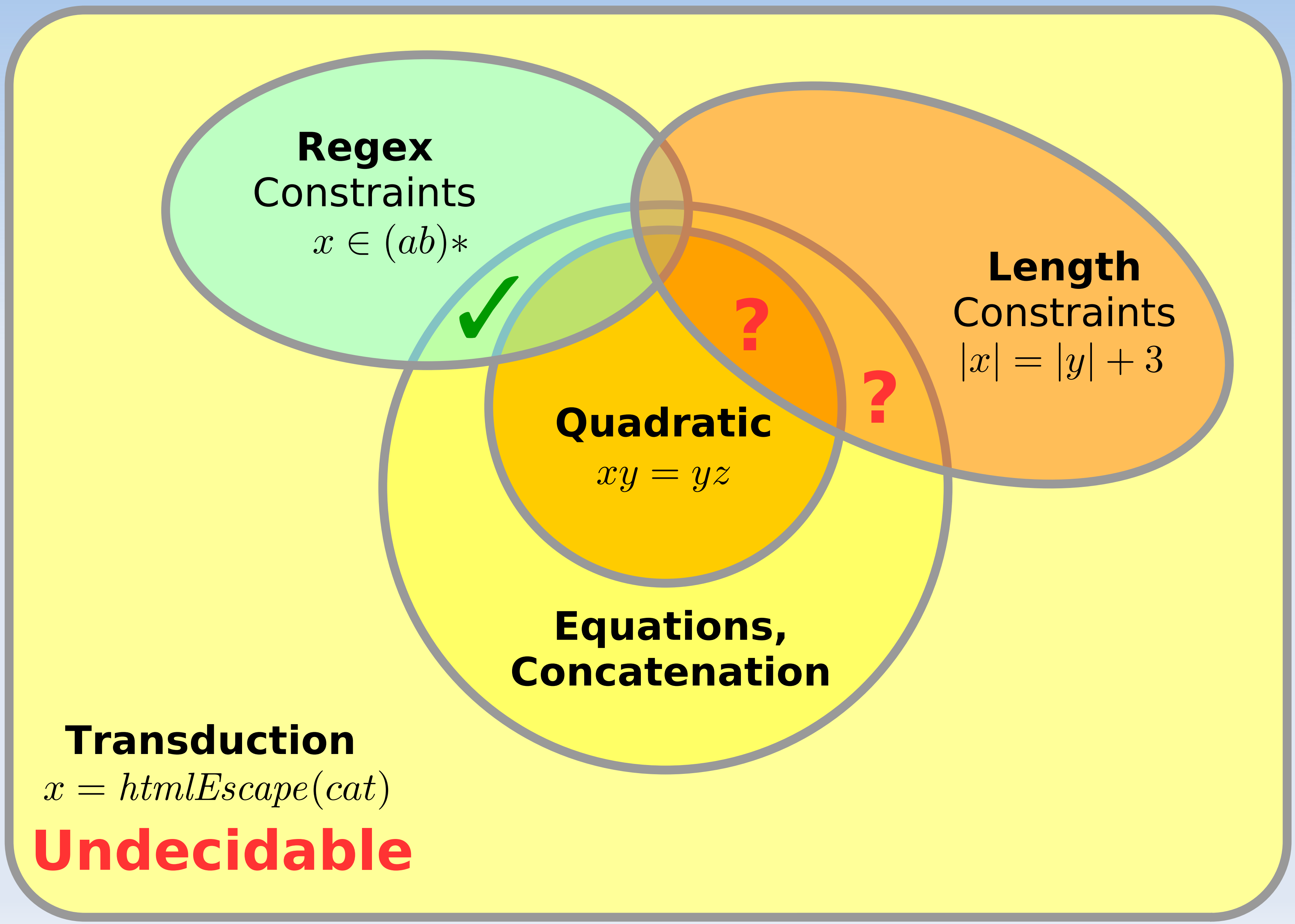
Combinations ...



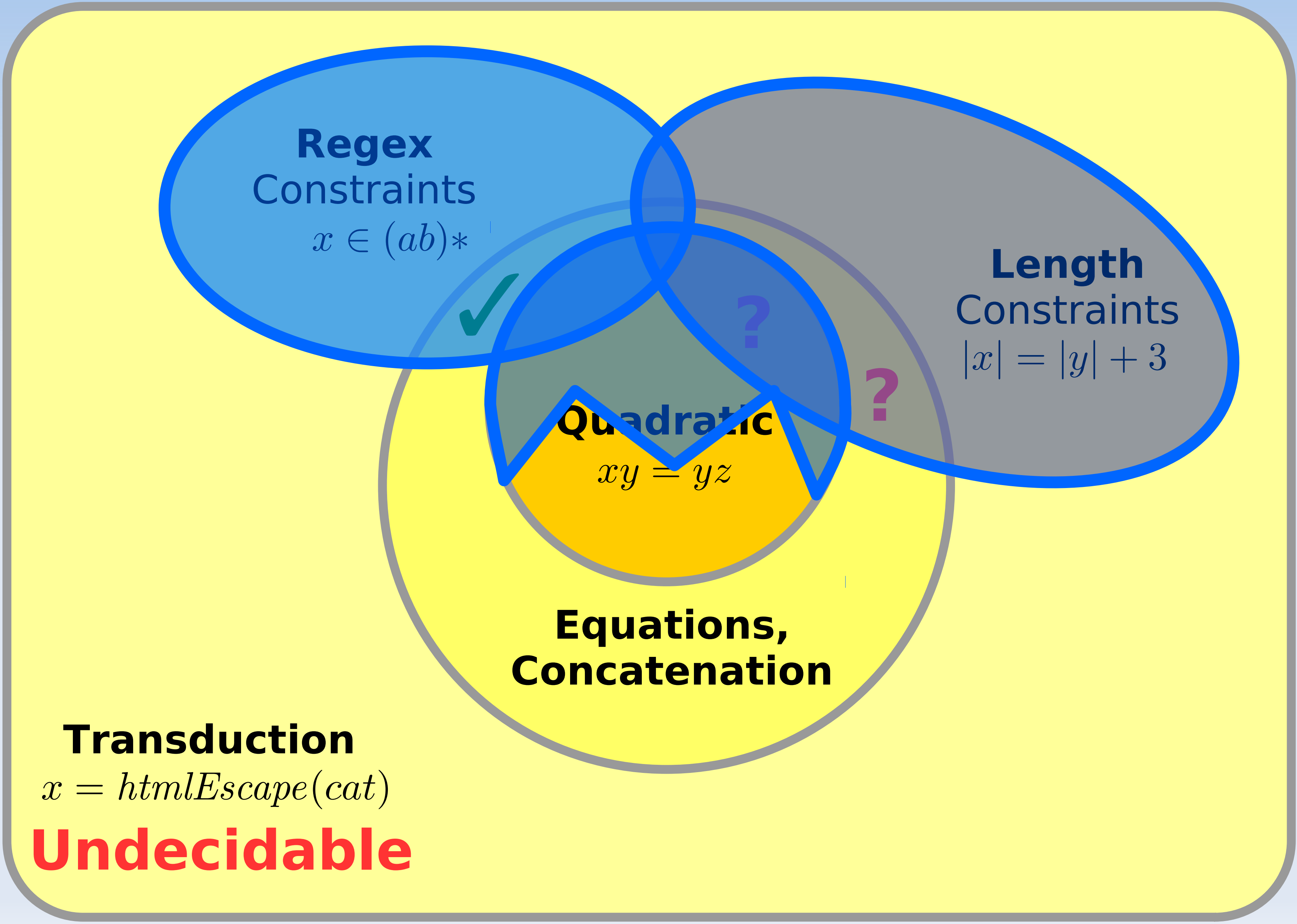
Combinations ...



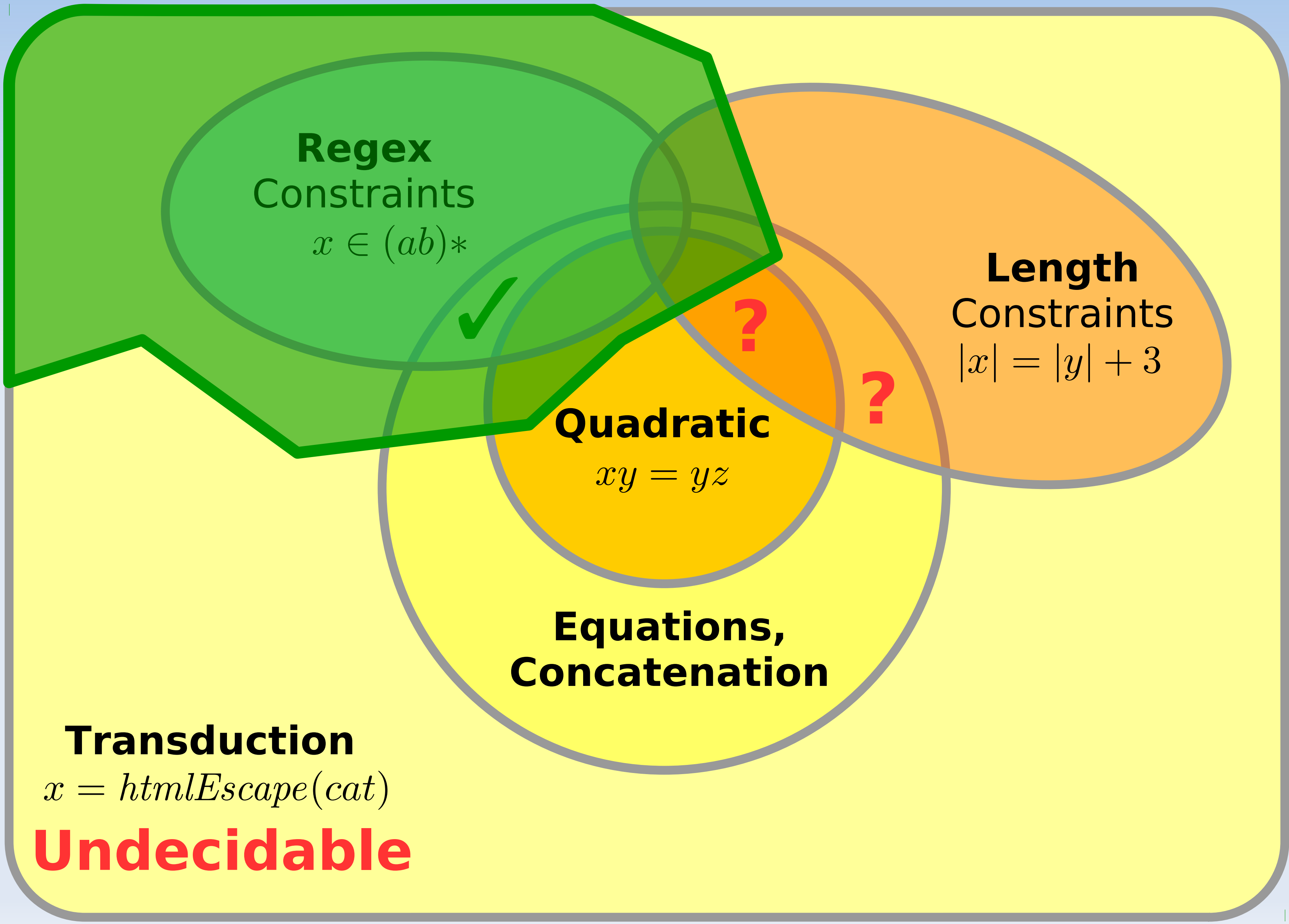
Combinations ...



Combinations …



Combinations ...



The Norn fragment

1. Boolean structure
2. Acyclic word equations
3. Regular expressions
4. Length constraints

Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, PR, Jari Stenman: **String Constraints for Verification**. CAV 2014

The Norn fragment

1. Boolean structure
2. Acyclic word equations
3. Regular expressions
4. Length constraints

Decidable!

Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, PR, Jari Stenman: **String Constraints for Verification**. CAV 2014

Examples

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

The Norn fragment

1. Boolean structure
   → DPLL/CDCL
2. Acyclic word equations
   → Splitting, Nielsen's trans.
3. Regular expressions
   → Automaton splitting
4. Length constraints
   → Length abstraction

Order in which procedure handles operators

Norn On One Slide

$xy = u\,\text{'b'}\,v \quad \wedge \quad y \in (ab)^* \quad \wedge \quad |x| = |y|$

Word Equations

Regular Expressions

Length Constraints

Norn On One Slide

$xy = u\,'b'\,v \quad \wedge \quad y \in (ab)^* \quad \wedge \quad |x| = |y|$

Norn On One Slide

$$xy = u\,\text{'b'}\,v \quad \wedge \quad y \in (ab)^* \quad \wedge \quad |x| = |y|$$

$$x = u_1$$

$$\wedge$$

$$y = u_2\,\text{'b'}\,v$$

$$\wedge$$

$$u = u_1 u_2$$

Norn On One Slide

$$xy = u\,\text{'b'}\,v \quad \wedge \quad y \in (ab)^* \quad \wedge \quad |x| = |y|$$

$$x = u_1$$

$$\wedge$$

$$y = u_2\,\text{'b'}\,v$$

$$\wedge$$

$$u = u_1 u_2$$

Norn On One Slide

$$xy = u\,\text{'b'}\,v \;\wedge\; y \in (ab)^* \;\wedge\; |x| = |y|$$

$$x = u_1$$

$$\wedge$$

$$y = u_2\,\text{'b'}\,v$$

$$\wedge$$

$$u = u_1 u_2$$

$$u_2\,\text{'b'}\,v \in (ab)^*$$

Norn On One Slide

$$xy = u\,'b'\,v \;\wedge\; y \in (ab)^* \;\wedge\; |x| = |y|$$

$$x = u_1$$

$$\wedge$$

$$y = u_2\,'b'\,v$$

$$\wedge$$

$$u = u_1 u_2$$

$$u_2\,'b'\,v \in (ab)^*$$

$$|u_1| = |u_2\,'b'\,v|$$
$$= |u_2| + 1 + |v|$$

Norn On One Slide

$$xy = u\,'b'\,v \quad \wedge \quad y \in (ab)^* \quad \wedge \quad |x| = |y|$$

$$x = u_1$$

$$\wedge$$

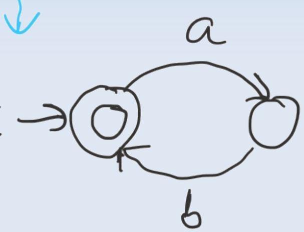
$$y = u_2\,'b'\,v$$

$$\wedge$$

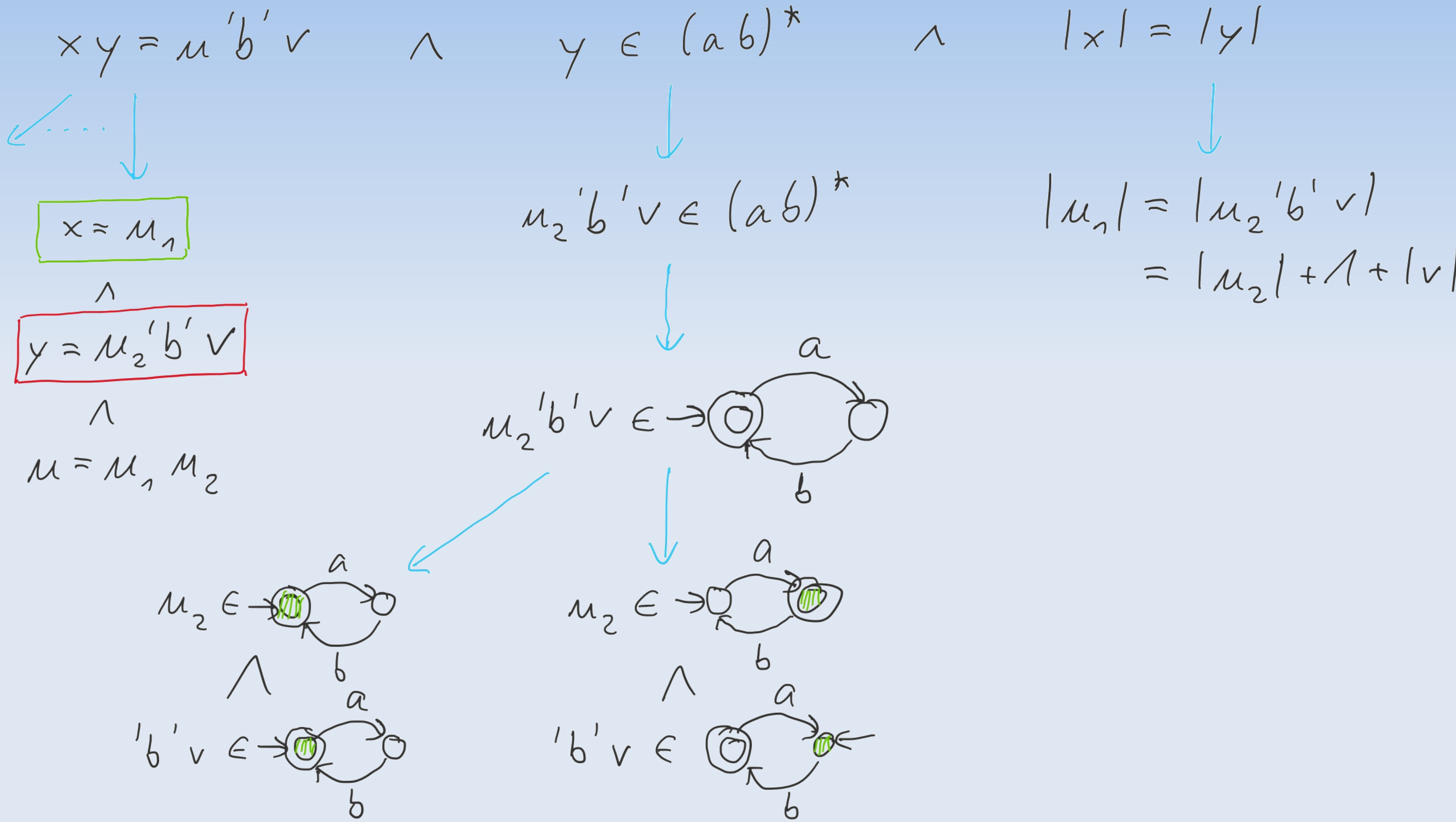
$$u = u_1 u_2$$

$$u_2\,'b'\,v \in (ab)^*$$

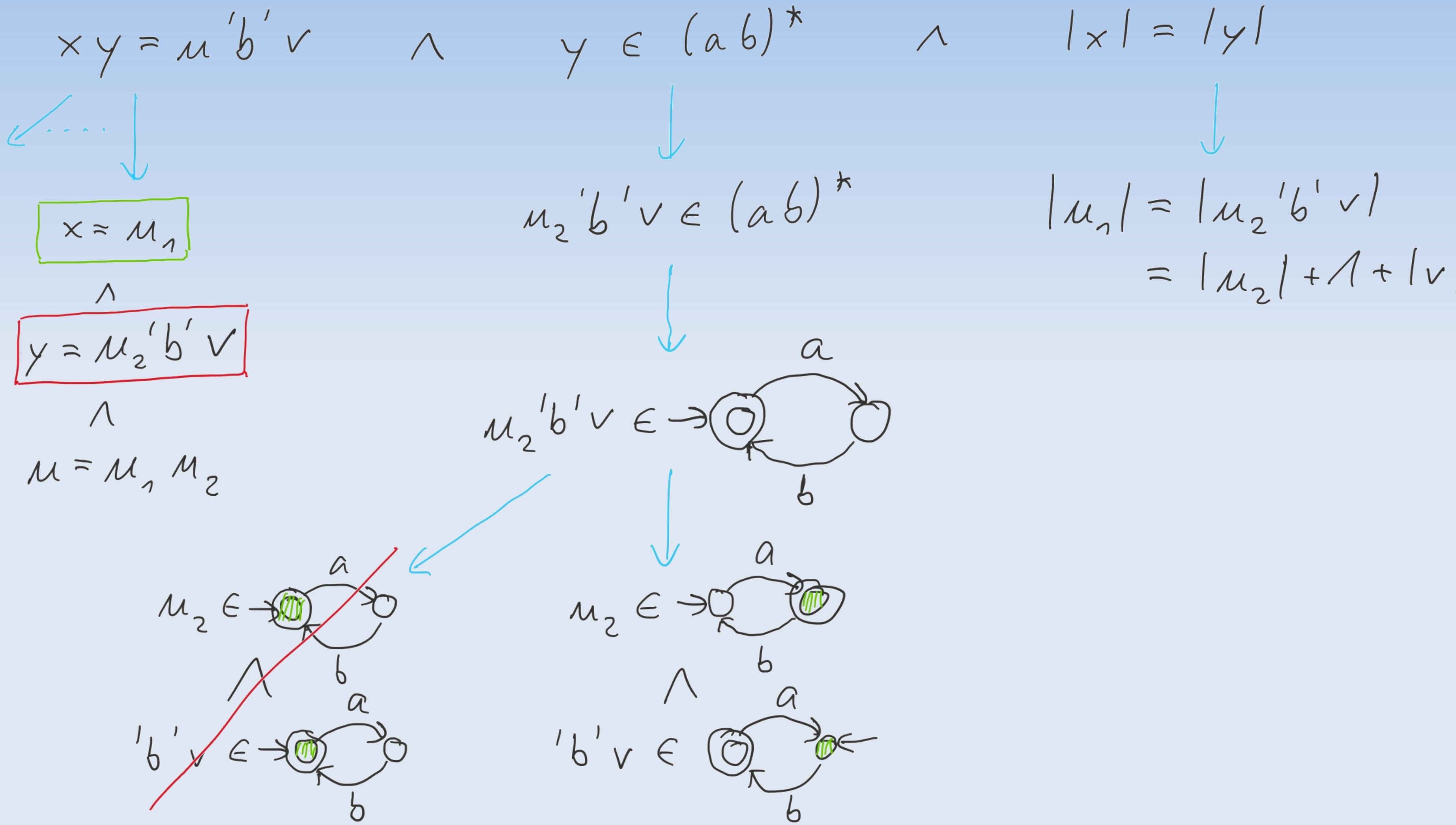
$$|u_1| = |u_2\,'b'\,v| = |u_2| + 1 + |v|$$

$u_2\,'b'\,v \in$ 

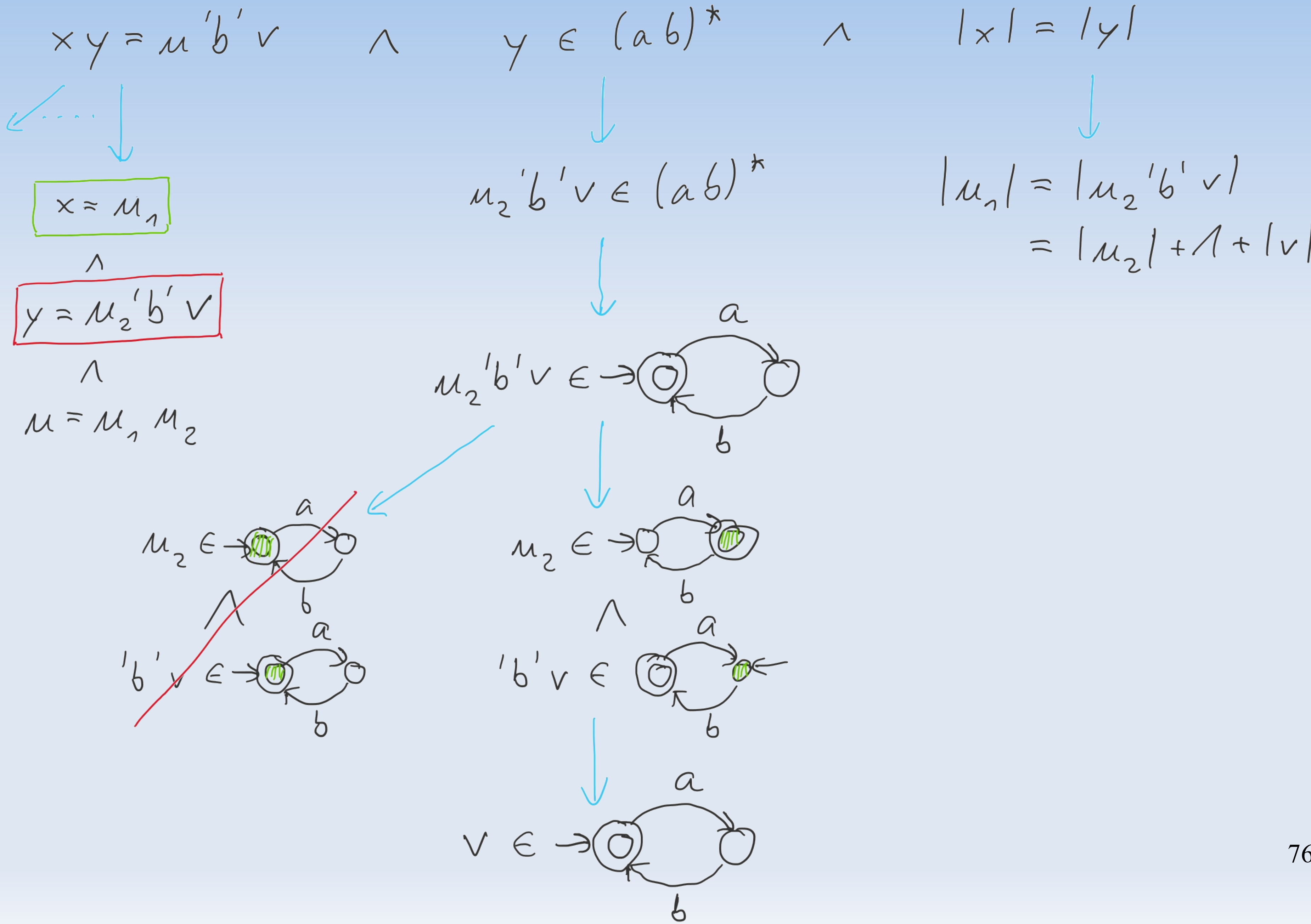
Norn On One Slide



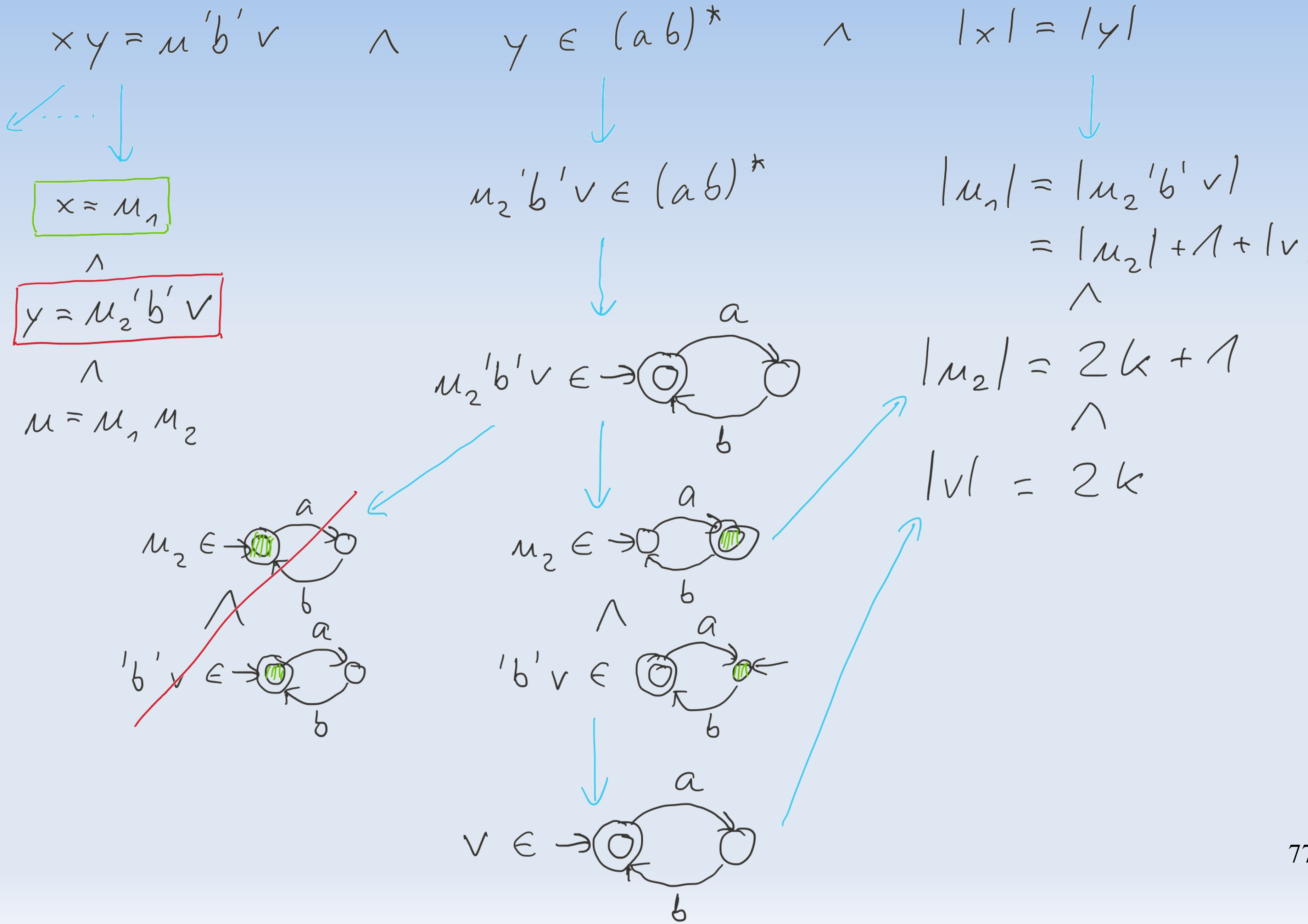
Norn On One Slide



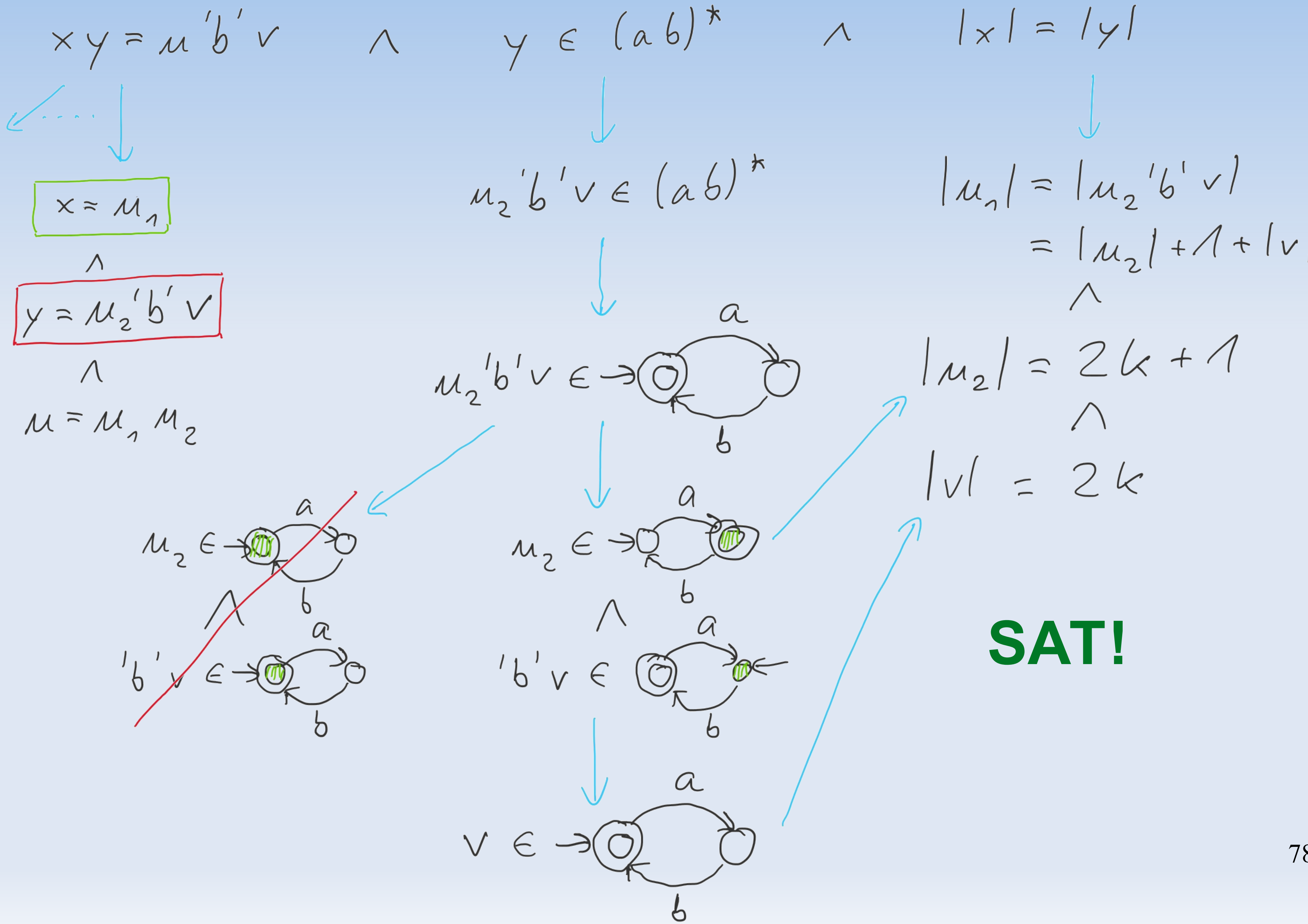
Norn On One Slide



Norn On One Slide



Norn On One Slide



The Norn fragment

1. Boolean structure
   → DPLL/CDCL
2. Acyclic word equations
   → Splitting, Nielsen's trans.
3. Regular expressions
   → Automaton splitting
4. Length constraints
   → Length abstraction
5. (Optimisations)

Order in which procedure handles operators

3. Splitting Automata

Lemma

Let $\mathcal{L}$ be a regular language. Then there are languages

$$\{(\mathcal{L}_1^i, \mathcal{L}_2^i)\}_{i=1}^n$$

such that

$$x \cdot y \in \mathcal{L} \quad \Leftrightarrow \quad \bigvee_{i=1}^{n} x \in \mathcal{L}_1^i \wedge y \in \mathcal{L}_2^i$$

3. Splitting Automata

Lemma
Let $\mathcal{L}$ be a regular language. Then there are languages

$$\{(\mathcal{L}_1^i, \mathcal{L}_2^i)\}_{i=1}^n$$

such that

$$x \cdot y \in \mathcal{L} \quad \Leftrightarrow \quad \bigvee_{i=1}^{n} x \in \mathcal{L}_1^i \wedge y \in \mathcal{L}_2^i$$

Disjunction over states of automaton representing $\mathcal{L}$

4. Length constraints

Lemma

Let $\mathcal{L}$ be a regular language. Then the length abstraction

$$\{|w| \mid w \in \mathcal{L}\}$$

is semi-linear (= Presburger-definable).

4. Length constraints

Lemma

Let $\mathcal{L}$ be a regular language. Then the length abstraction

$$\{|w| \mid w \in \mathcal{L}\}$$

is semi-linear (= Presburger-definable).

Special case of the Parikh image of a regular language

Back to the Wishlist

- Concatenation, splitting
- Length
- Regular expressions
- Search/replace
- Conversions: strings ↔ numbers
- Encoding/decoding, escaping

- ASCII (7/8 bit), Unicode (21 bit)
- Representations: UTF-8, UTF-16, UTF-32

Back to the Wishlist

- Concatenation, splitting ✓
- Length ✓
- Regular expressions ✓
- Search/replace
- Conversions: strings ↔ numbers
- Encoding/decoding, escaping

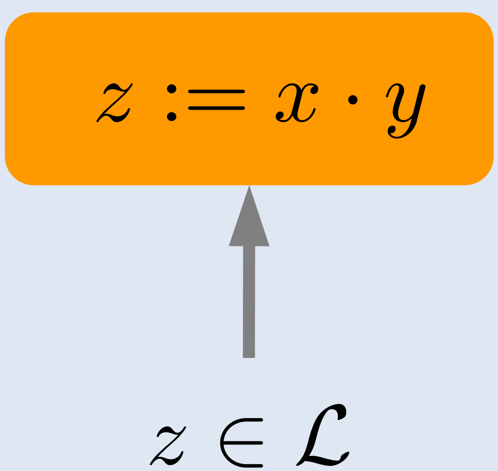
- ASCII (7/8 bit), Unicode (21 bit) ✓ ✓
- Representations: UTF-8, UTF-16, UTF-32


Starting from
the End …

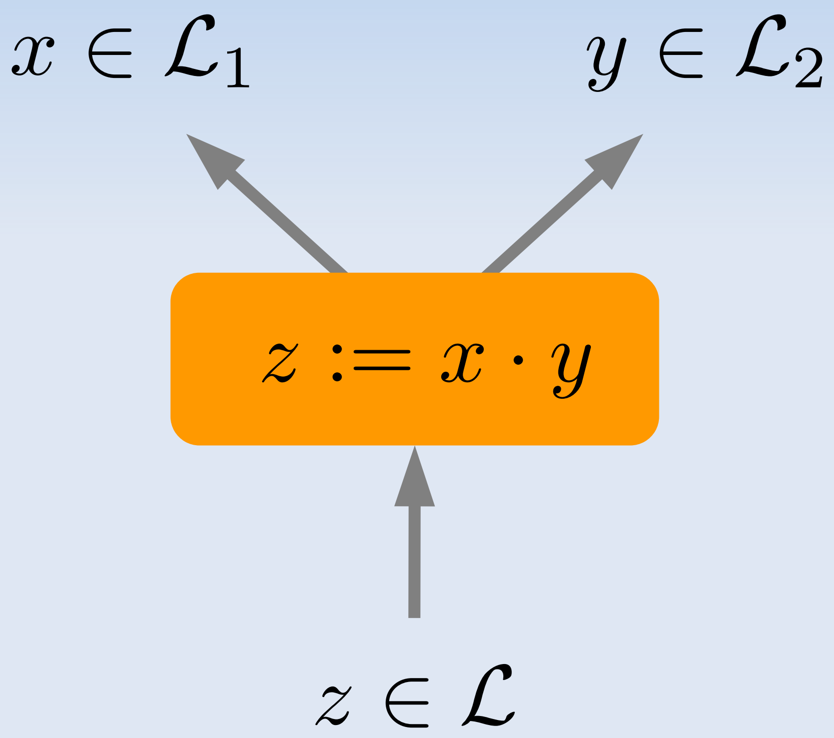
Backward-Propagation

$$z := x \cdot y$$

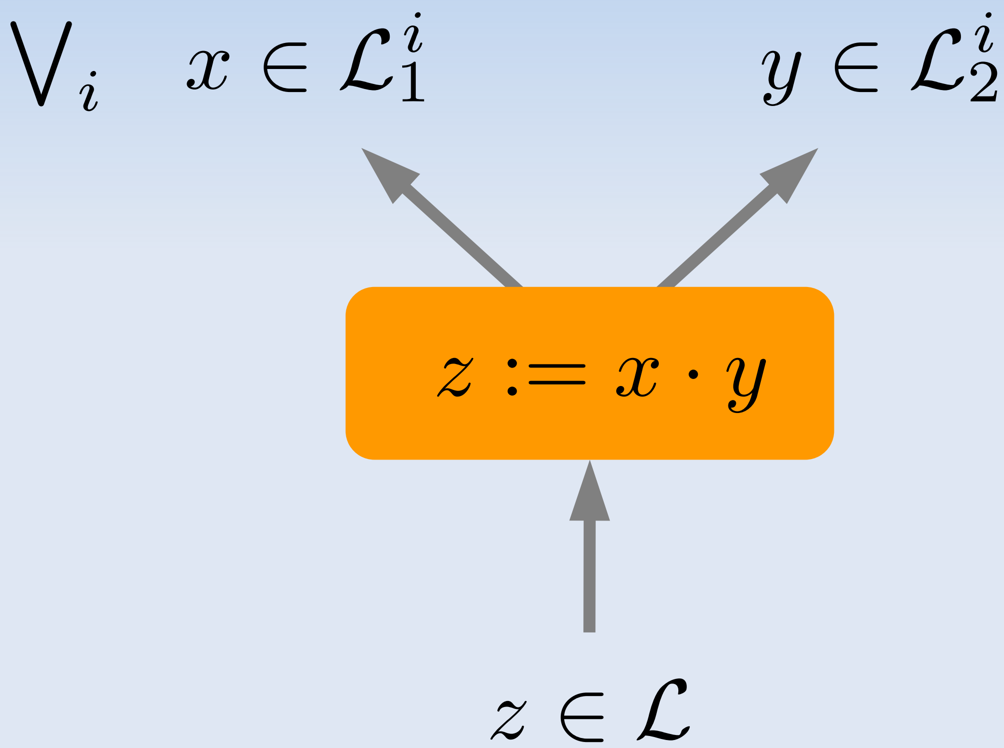
Backward-Propagation



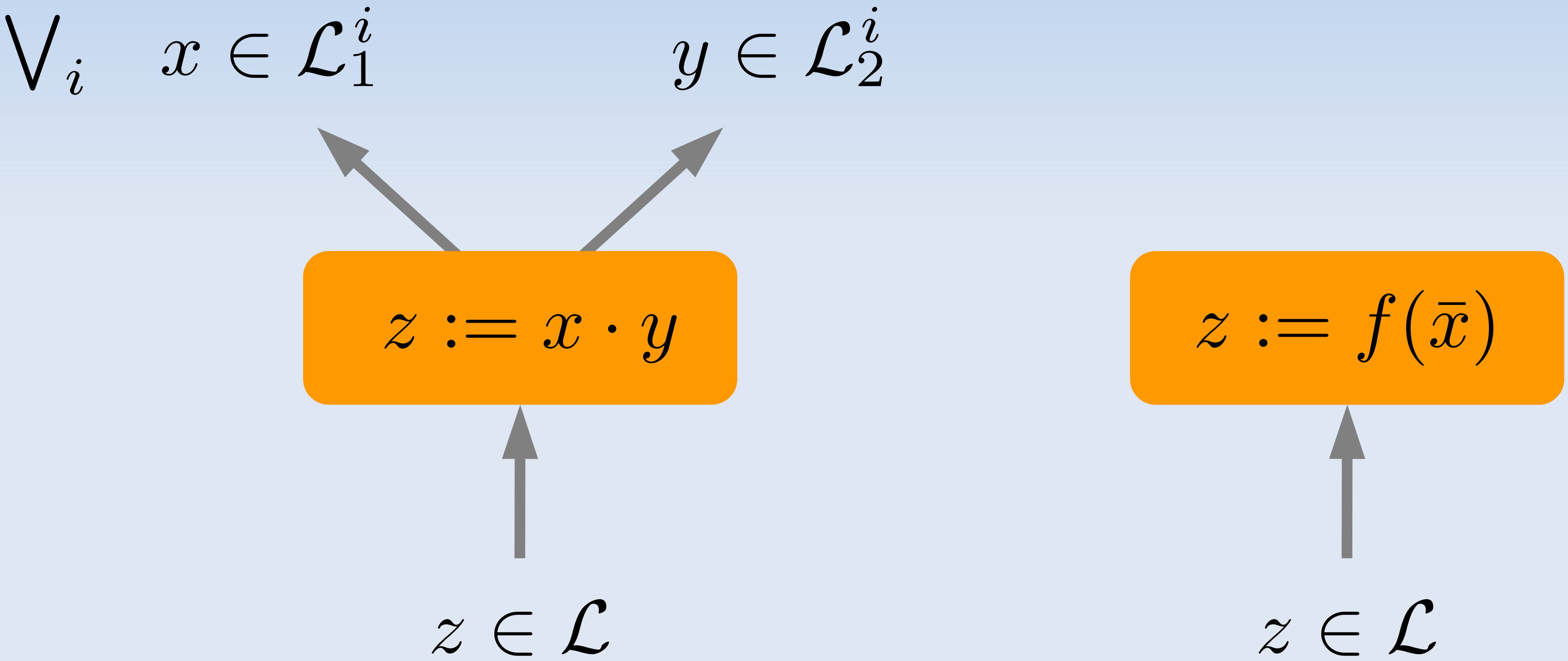
Backward-Propagation



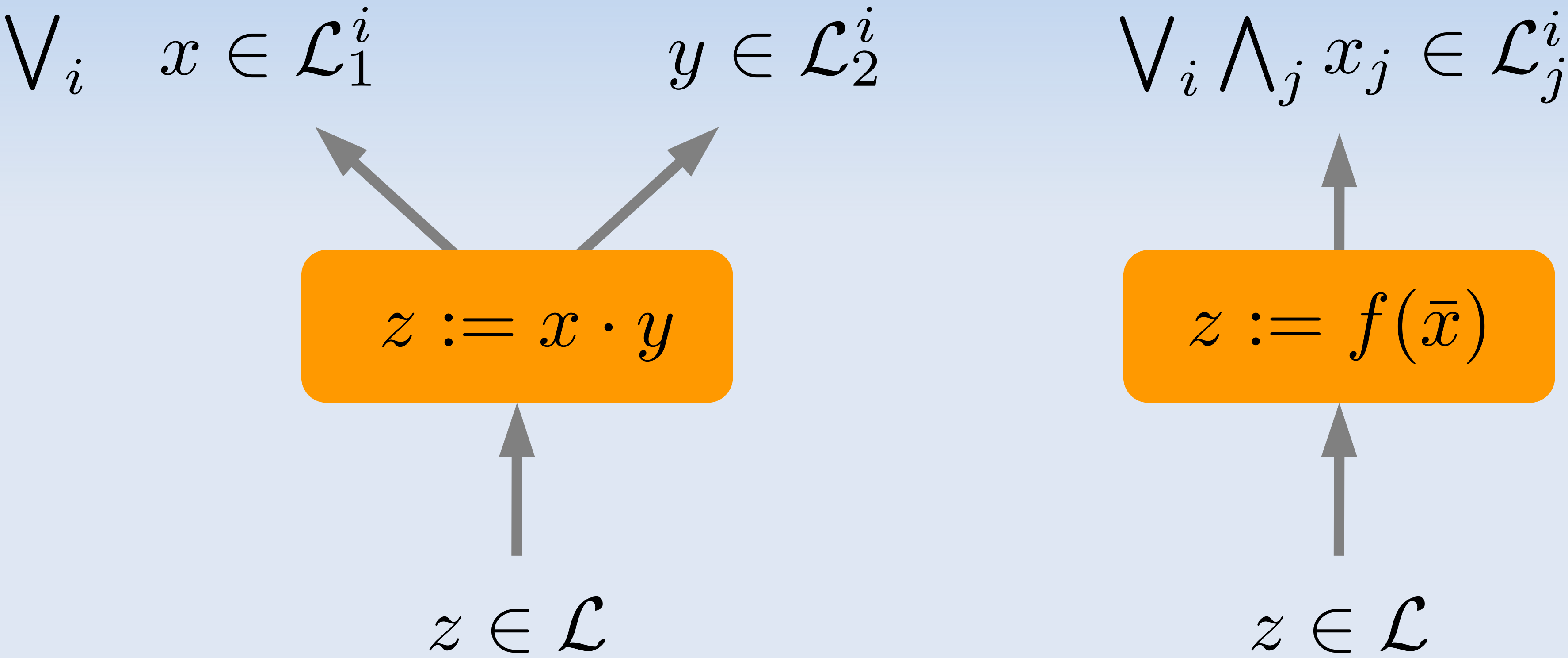
Backward-Propagation



Backward-Propagation



Backward-Propagation



3. Splitting Automata

Lemma

Let $\mathcal{L}$ be a regular language. Then there are languages

$$\{(\mathcal{L}_1^i, \mathcal{L}_2^i)\}_{i=1}^n$$

such that

$$x \cdot y \in \mathcal{L} \quad \Leftrightarrow \quad \bigvee_{i=1}^{n} x \in \mathcal{L}_1^i \wedge y \in \mathcal{L}_2^i$$

3. Splitting Automata

Lemma

Let $\mathcal{L}$ be a regular language. Then there are languages

$$\{(\mathcal{L}_1^i, \mathcal{L}_2^i)\}_{i=1}^n$$

such that

$$x \cdot y \in \mathcal{L} \quad \Leftrightarrow \quad \bigvee_{i=1}^{n} x \in \mathcal{L}_1^i \wedge y \in \mathcal{L}_2^i$$

Disjunction over states of automaton representing $\mathcal{L}$

The OSTRICH Fragment(s)

1. Straight-line formulas/programs with
   Regular expressions and
   Assignments
2. Admissible functions

Taolue Chen, Matthew Hague, Anthony W. Lin, PR, Zhilin Wu: **Decision procedures for path feasibility of string-manipulating programs with complex operations**. PACMPL 3(POPL): 49:1-49:30 (2019)

The OSTRICH Fragment(s)

1. Straight-line formulas/programs with
   Regular expressions and
   Assignments
2. Admissible functions

(Also decidable!)

Taolue Chen, Matthew Hague, Anthony W. Lin, PR, Zhilin Wu: **Decision procedures for path feasibility of string-manipulating programs with complex operations**. PACMPL 3(POPL): 49:1-49:30 (2019)

Straight-line Formulas

- Conjunctions of constraints sorted by dependency:

$$\phi_1 \wedge \cdots \wedge \phi_n$$

- Each ϕ_i is $x_i := f(\bar{y})$ or $x \in \mathcal{L}$
- All x_i pairwise distinct
- Each x_i may only occur in $\phi_{i+1}, \ldots, \phi_n$

- Close to programs in SSA form

Example

x in a*b*

y := reverse(x)

y in b*a*

z := replaceAll(y, a, b)

z in b*

Is there an input x satisfying all assertions?

Example

x in a*b*

y := reverse(x)

y in b*a*

z := replaceAll(y, a, b)

z in b*

} y in (a | b)*

Is there an input x satisfying all assertions?

Example

x in a*b*

y := reverse(x)

y in b*a*

y in (a | b)*

Is there an input x satisfying all assertions?

Example

x in a*b*

y := reverse(x)

y in b*a*

y in (a | b)*

} y in (a | b)* & b*a*

Is there an input x satisfying all assertions?

Example

x in a*b*

y := reverse(x)

y in b*a*

Is there an input x satisfying all assertions?

Example

x in a*b*

y := reverse(x)

y in b*a*

} x in a*b*

Is there an input x satisfying all assertions?

Example

x in a*b*

x in a*b*

Is there an input x satisfying all assertions?

Example

x in a*b*

Is there an input x satisfying all assertions?

Example

Easy to solve!

x in a*b*

Is there an input x satisfying all assertions?

Admissible Functions

Definition
A function

$$f : String \times \cdots \times String \to String$$

is **admissible** if the pre-image $f^{-1}(\mathcal{L})$ of every regular language $\mathcal{L}$ can be represented in the form

$$f^{-1}(\mathcal{L}) = \bigcup_{i=1}^{n} \mathcal{L}_1^i \times \cdots \times \mathcal{L}_k^i$$

for regular languages

$$\{(\mathcal{L}_1^i, \ldots, \mathcal{L}_k^i)\}_{i=1}^{n}$$

Admissible Functions

Definition

A function

$$f : String \times \cdots \times String \rightarrow String$$

is **admissible** if the pre-image $f^{-1}(\mathcal{L})$ of every regular language $\mathcal{L}$ can represented in the form

$$f^{-1}(\mathcal{L}) = \bigcup_{i=1}^{n} \mathcal{L}_1^i \times \cdots \times \mathcal{L}_k^i$$

for regular languages

$$\{(\mathcal{L}_1^i, \ldots, \mathcal{L}_k^i)\}_{i=1}^{n}$$

Pre-image is **recognisable**

Some Admissible Functions

- Concatenation
- Replace, replace-all

  (even if replacement string is a variable)
- Reverse
- Unary functions defined by transducers
- Conversions: UTF-8, UTF-16, UTF-32, etc.

A Non-Admissible Function

- The function converting from unary to binary number representation
  - Pre-image of a regular language is in general not regular

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$y \in \mathcal{L}_2$$

The OSTRICH Algorithm

Function applications

$y := f(x)$

$z := g(y)$

Active regexes

$x \in \mathcal{L}_1$

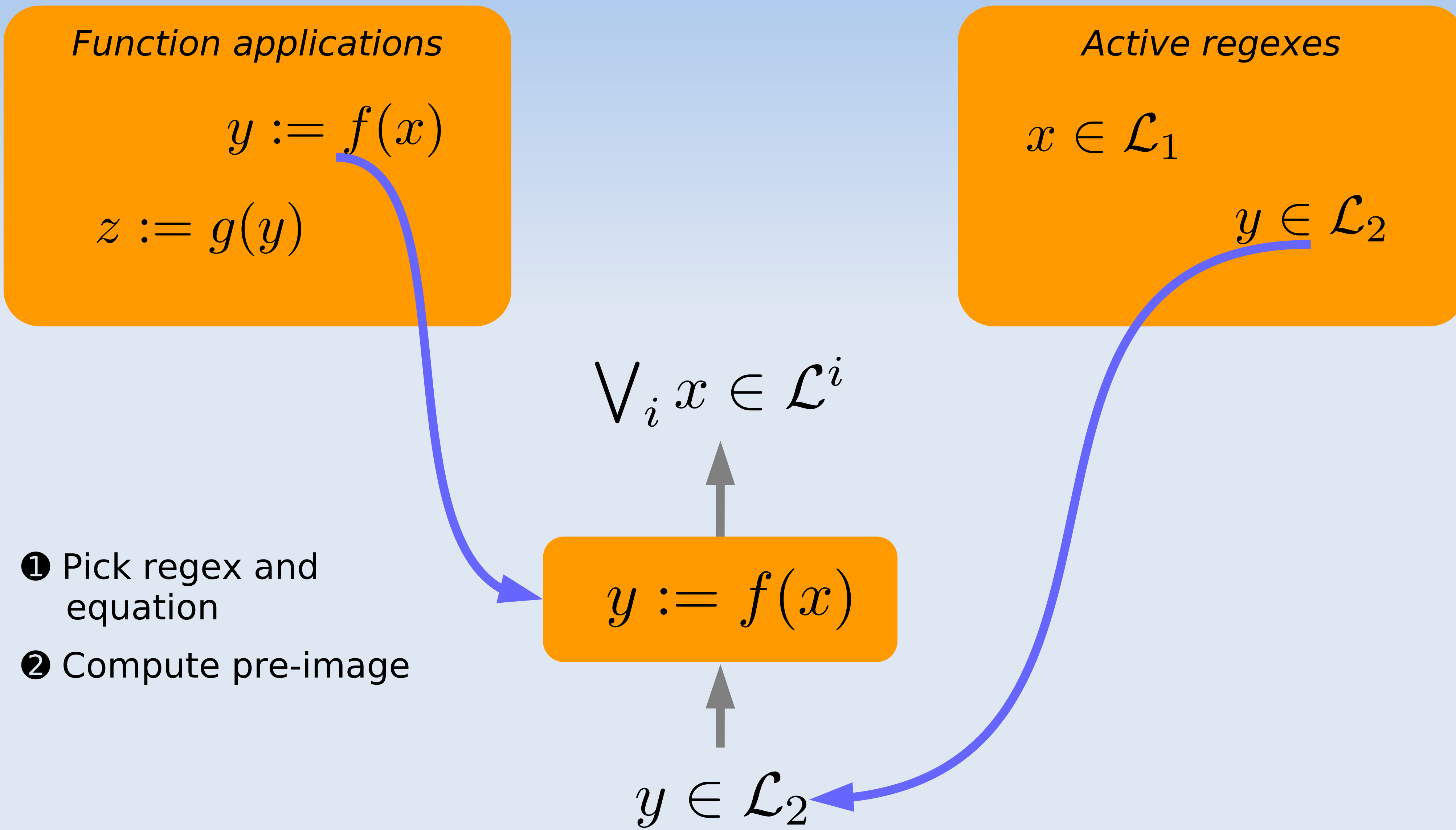
$y \in \mathcal{L}_2$

➊ Pick regex and equation

$y := f(x)$

$y \in \mathcal{L}_2$

The OSTRICH Algorithm



❶ Pick regex and equation

❷ Compute pre-image

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$\bigvee_i x \in \mathcal{L}^i$$

❶ Pick regex and equation

❷ Compute pre-image

Passive regexes

$$y \in \mathcal{L}_2$$

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$x \in \mathcal{L}^i$$

$$\bigvee_i x \in \mathcal{L}^i$$

Passive regexes

$$y \in \mathcal{L}_2$$

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$x \in \mathcal{L}^i$$

Passive regexes

$$y \in \mathcal{L}_2$$

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

❹ Check regex consistency

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$x \in \mathcal{L}^i$$

Passive regexes

$$y \in \mathcal{L}_2$$

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

❹ Check regex consistency

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

$$x \in \mathcal{L}^i$$

Passive regexes

$$y \in \mathcal{L}_2$$

➊ Pick regex and equation

➋ Compute pre-image

➌ Recurse over new regexes

➍ Check regex consistency

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

...

$$\bigvee_i x \in \mathcal{L}^i$$

Passive regexes

$$y \in \mathcal{L}_2$$

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

❹ Check regex consistency

❺ Compute conflict set + back-jump

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

...

$$\bigvee_i x \in \mathcal{L}^i$$

Intersection of regular languages, product automata

ssive regexes

$$y \in \mathcal{L}_2$$

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

❹ Check regex consistency

❺ Compute conflict set + back-jump

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

...

$$\bigvee_i x \in \mathcal{L}^i$$

Intersection of regular languages, product automata

ssive regexes

Greedy computation of minimal conflict set

❶ Pick regex and equation

❷ Compute pre-image

❸ Recurse over new regexes

❹ Check regex consistency

❺ Compute conflict set + back-jump

The OSTRICH Algorithm

Function applications

$$y := f(x)$$

$$z := g(y)$$

Active regexes

$$x \in \mathcal{L}_1$$

...

Automata computation

$\bigvee_i x \in \mathcal{L}^i$

Intersection of regular languages, product automata

...ssive regexes

Greedy computation of minimal conflict set

➊ Pick regex and equation

➋ Compute pre-image

➌ Recurse over new regexes

➍ Check regex consistency

➎ Compute conflict set + back-jump

Pre-images?

- Concatenation
  - Splitting over automata states (like in Norn)
- Replace, replace-all
  - Reduction to transducers; Cayley graphs
- Reverse
  - Easy: revert automata transitions
- Unary functions defined by transducers
  - Product construction, projection

In the Example

JavaScript embedded in a web-page

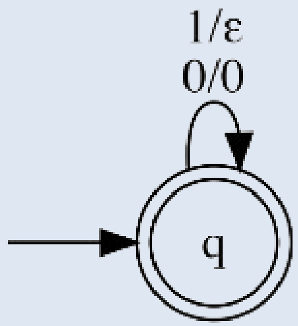
```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

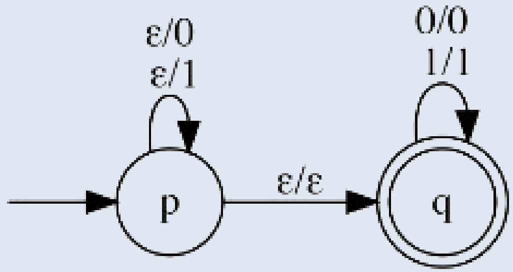
$$
\begin{aligned}
& x = htmlEscape(cat) \\
\wedge\ & y = escape(x) \\
\wedge\ & z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3 \\
\wedge\ & innerHTML = htmlUnescape(z) \\
\wedge\ & attack(innerHTML)
\end{aligned}
$$

Transducers

Definition
An ***n*-track transducer** is a finite-state automaton over the alphabet Σ_ϵ^n

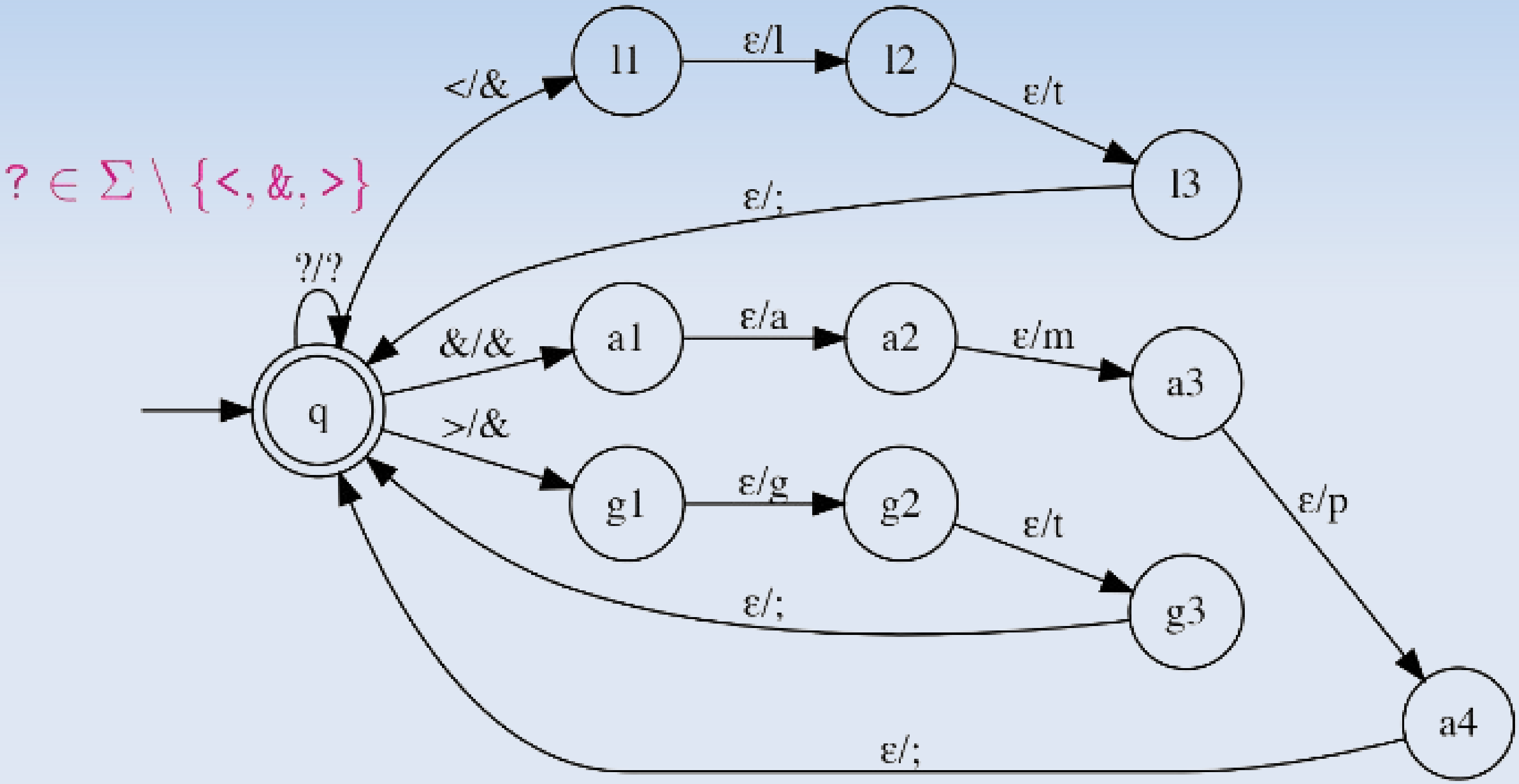


Erase all occurrences of 1



Input is a suffix of output

HTML Escaping



Replace: < by <, > by >, and & by &

In the Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$
\begin{aligned}
& x = htmlEscape(cat) \\
\wedge\ & y = escape(x) \\
\wedge\ & z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3 \\
\wedge\ & innerHTML = htmlUnescape(z) \\
\wedge\ & attack(innerHTML)
\end{aligned}
$$

In the Example

JavaScript embedded in a web-page

```
var x = goog.string.htmlEscape(cat);
var y = goog.string.escapeString(x);

catElem.innerHTML =
  '<button onclick="createCatList(\'' +
  y + '\')">' + x + '</button>';
```

$$x = htmlEscape(cat)$$

$$\wedge\ y = escape(x)$$

Proven **SAT** by OSTRICH (in a few seconds)

$$\wedge\ innerHTML = htmlUnescape(z)$$

$$\wedge\ attack(innerHTML)$$

Extensions

- More general word equations
  - E.g., acyclic fragment
- Length, integer functions
  - `str.len`
  - `str.at`
  - `str.substr`
  - *etc.*

Summary

Norn + OSTRICH(+)

- Decision procedures for rich fragments of string constraints
  - Regular expressions, length
  - Acyclic word equations
  - Admission functions

- https://github.com/uuverifiers/ostrich

Summary

- String solving is an exciting area!
  - Many new ideas
  - Many challenges

- **Q1**: What is decidable about strings?
- **Q2**: What is the right string logic?
- **Q3**: How to build efficient solvers?

Thank you for your attention!

Vetenskapsrådet