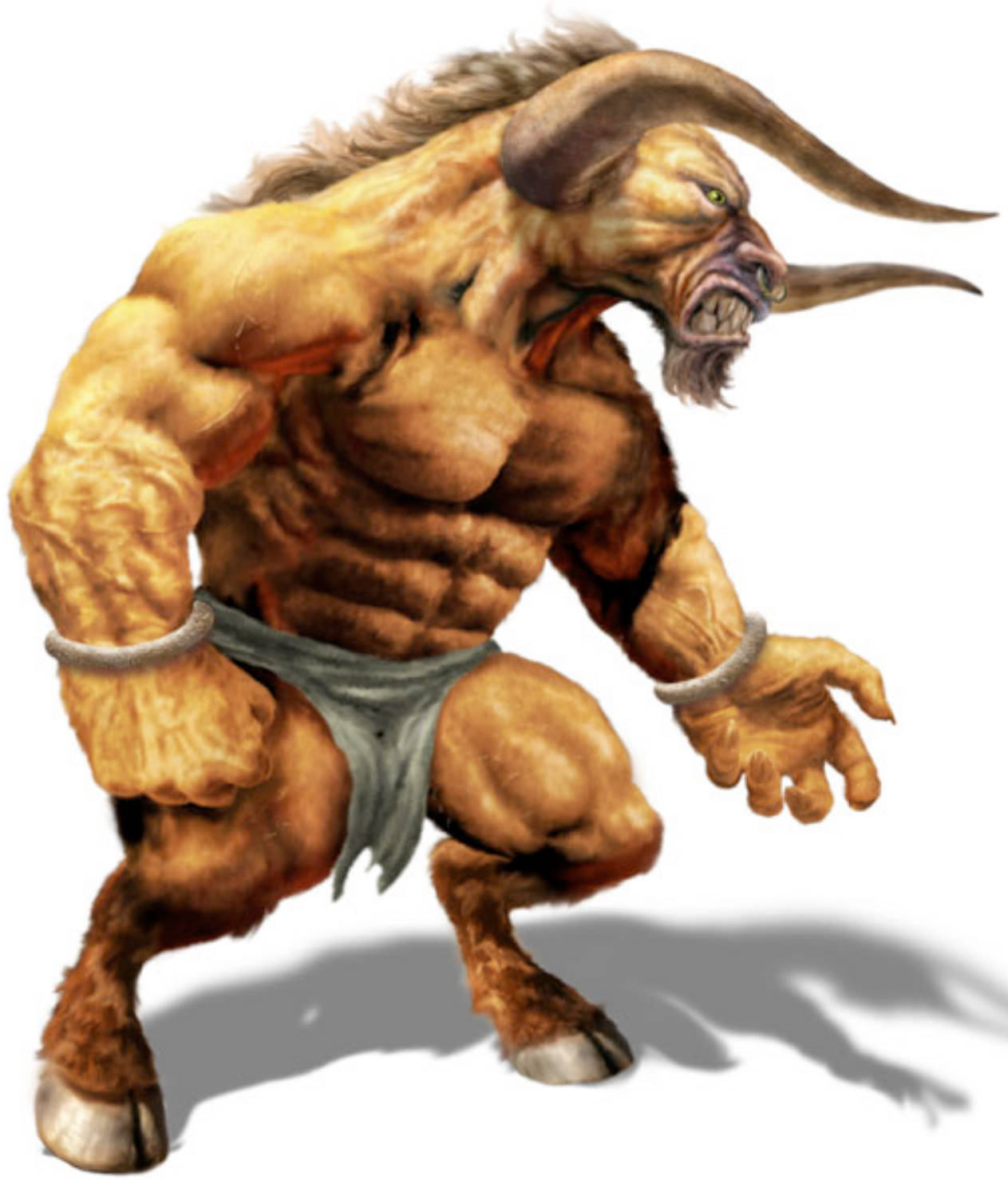
Lexical Analysis

Matt Might
University of Utah
matt.might.net



flex

Derivatives

Matt Might

University of Utah

matt.might.net

No office hours today

Live code

Today

- Derivatives of regular expressions
- How to match regular expressions

1964

Derivatives of Regular Expressions

JANUSZ A. BRZOZOWSKI

Princeton University, Princeton, New Jersey†

Abstract. Kleene's regular expressions, which can be used for describing sequential circuits, were defined using three operators (union, concatenation and iterate) on sets of sequences. Word descriptions of problems can be more easily put in the regular expression language if the language is enriched by the inclusion of other logical operations. However, in the problem of converting the regular expression description to a state diagram, the existing methods either cannot handle expressions with additional operators, or are made quite complicated by the presence of such operators. In this paper the notion of a derivative of a regular expression is introduced and the properties of derivatives are discussed. This leads, in a very natural way, to the construction of a state diagram from a regular expression containing any number of logical operators.

$$D_c L$$

1. **Filter**:
   Keep every string starting with c.

2. **Chop**:
   Remove c from the start of each.

$$D_{\mathfrak{f}}$$

foo

frak bar

$$D_c L = \{w : cw \in L\}$$

$$cw \in L \text{ iff } w \in D_c(L).$$

Recognition algorithm

- Derive with respect to each character.
- Does the derived language contain ε?

$$\texttt{foo} \in (\texttt{foo})*$$

$$\texttt{oo} \in D_{\texttt{f}}(\texttt{foo})*$$

*(OOオ)OO ∋ ƷO

$$\varepsilon \in (\mathtt{foo})*$$

Deriving atomic languages

$$\epsilon \equiv \{\text{''''}\}$$

$$c \equiv \{c\}$$

$$\emptyset \equiv \{\}$$

$$D_c\emptyset =$$

$$D_c(\epsilon) =$$

$$D_c\{c\} =$$

Deriving regular languages

$$L_1 \cup L_2$$

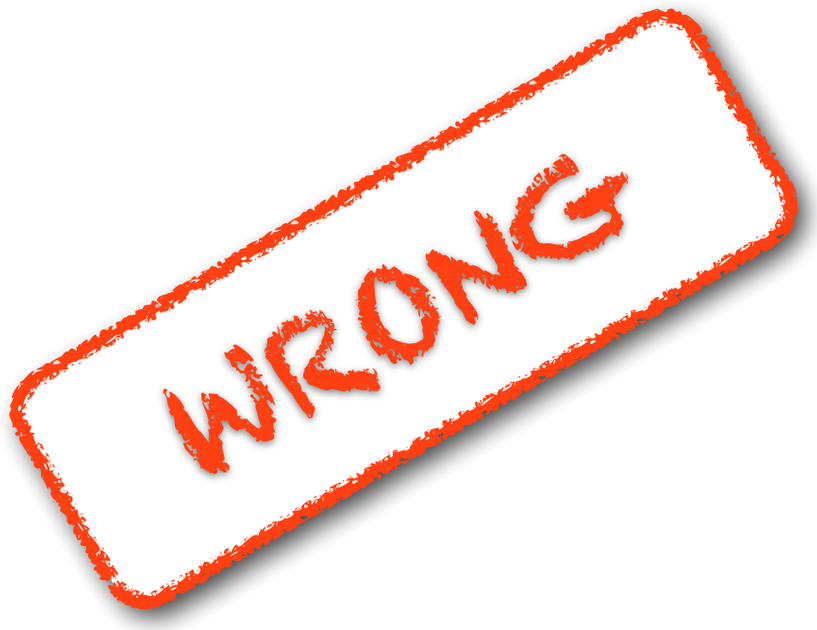
$$L_1 \cdot L_2$$

$$L_1^\star$$

$$D_c(L_1 \cup L_2)$$

$$D_c(L^\star) =$$

Concatenation?



$$D_c(L_1 \cdot L_2) = (D_c L_1 \cdot L_2)$$

Needs nullability

$$\delta(L) = \epsilon \text{ if } \epsilon \in L$$

$$\delta(L) = \emptyset \text{ if } \epsilon \notin L$$

$$D_c(L_1 \cdot L_2) =$$

$$D_c(L_1 \cdot L_2) = (D_c L_1 \cdot L_2)$$

Suppose: $\delta(L_1) = \emptyset$

$$D_c(L_1 \cdot L_2) = (D_c L_1 \cdot L_2) \cup (\qquad\qquad D_c L_2)$$

Suppose: $\delta(L_1) = \epsilon$

To recognize?

Need nullability

Need to *compute* nullability

$$\delta(\epsilon) =$$

$$\delta(\epsilon) = \epsilon$$

$$\delta(c) =$$

$$\delta(c) = \emptyset$$

$$\delta(\emptyset) =$$

$$\delta(\emptyset) = \emptyset$$

$$\delta(L_1 \cup L_2) =$$

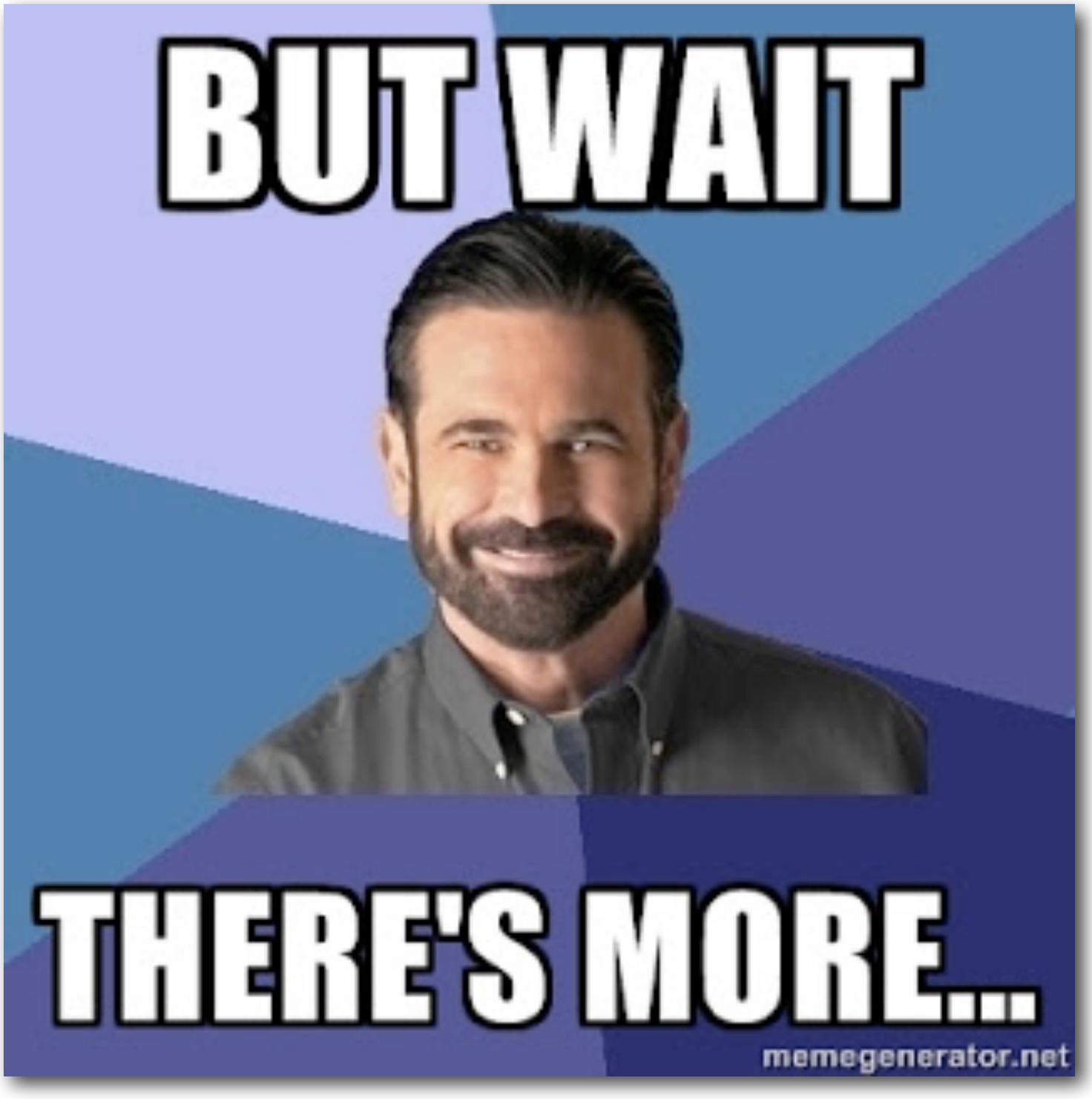
$$\delta(L_1 \cup L_2) = \delta(L_1) \cup \delta(L_2)$$

$$\delta(L_1 \cdot L_2) =$$

$$\delta(L_1 \cdot L_2) = \delta(L_1) \cdot \delta(L_2)$$

$$\delta(L_1^\star) =$$

$$\delta(L_1^\star) = \epsilon$$


BUT WAIT
THERE'S MORE...
memegenerator.net

$$D_c(L_1 \cap L_2) =$$

$$D_c(L_1 - L_2) =$$

$$D_c(\overline{L}) =$$

What about nullability?

$$\delta(L_1 \cap L_2) =$$

$$\delta(L_1 - L_2) =$$

$$\delta(\overline{L}) =$$

code

Lexing with derivatives

Algorithm

```
NaïveRemoveLongestMatch(w ∈ A*,R ⊆ 2^A*)
   suffix ← w
   while (w ≠ ε)
    if ∃L ∈ R : ε ∈ L
       suffix ← w
    c:w ← w
    R ← D_c.R
  return suffix
```

Algorithm

```
RemoveLongestMatch(w ∈ A*,R ⊆ 2^A*)
   suffix ← w
   while (R ≠ ∅ or w ≠ ε)
    if ∃L ∈ R : ε ∈ L
       suffix ← w
    c:w ← w
    R ← D_c.R - {∅}
  return suffix
```

Questions?