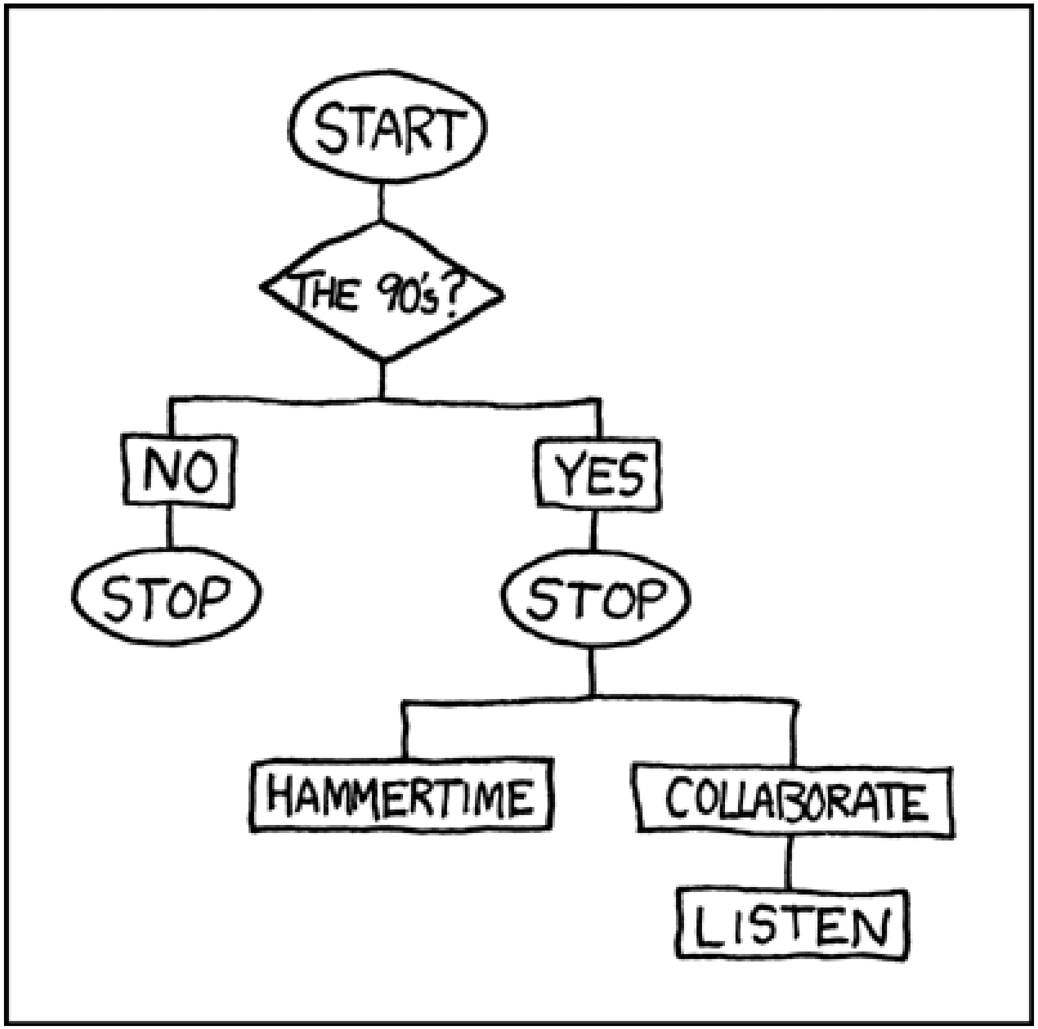
Indexed Languages

and why you care

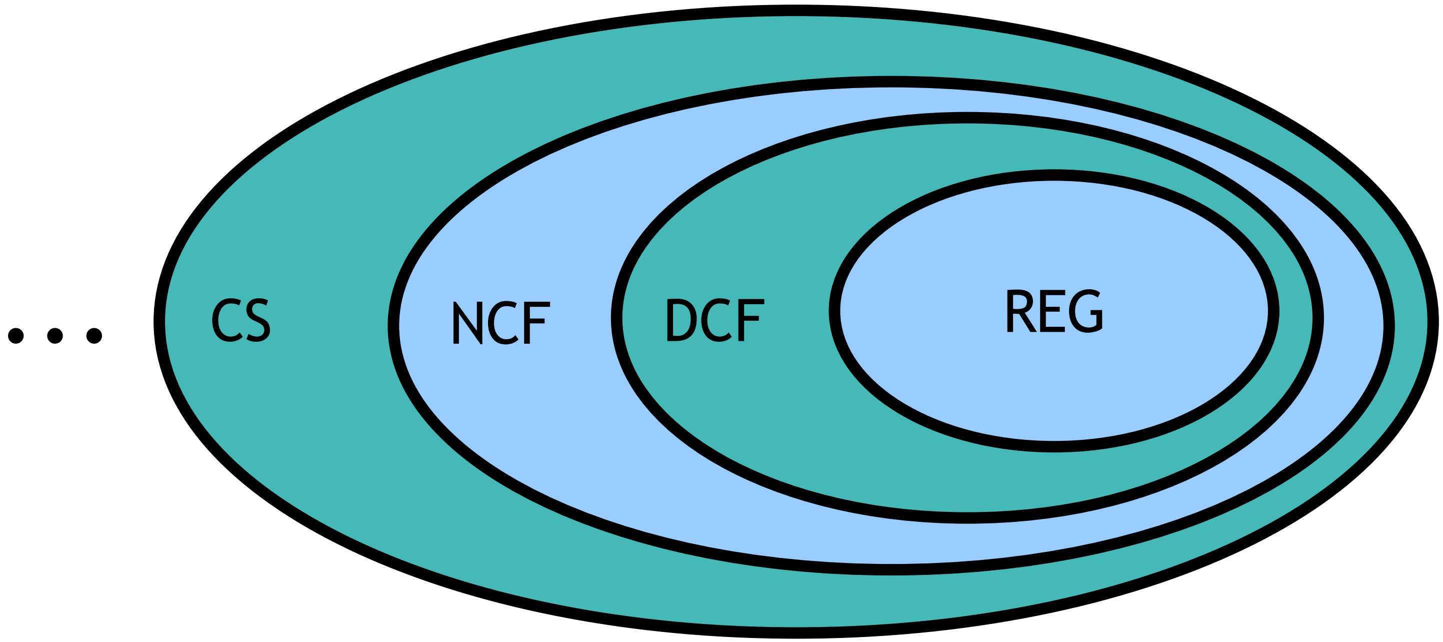
Presented by Pieter Hooimeijer
2008-02-07

Think Way Back...

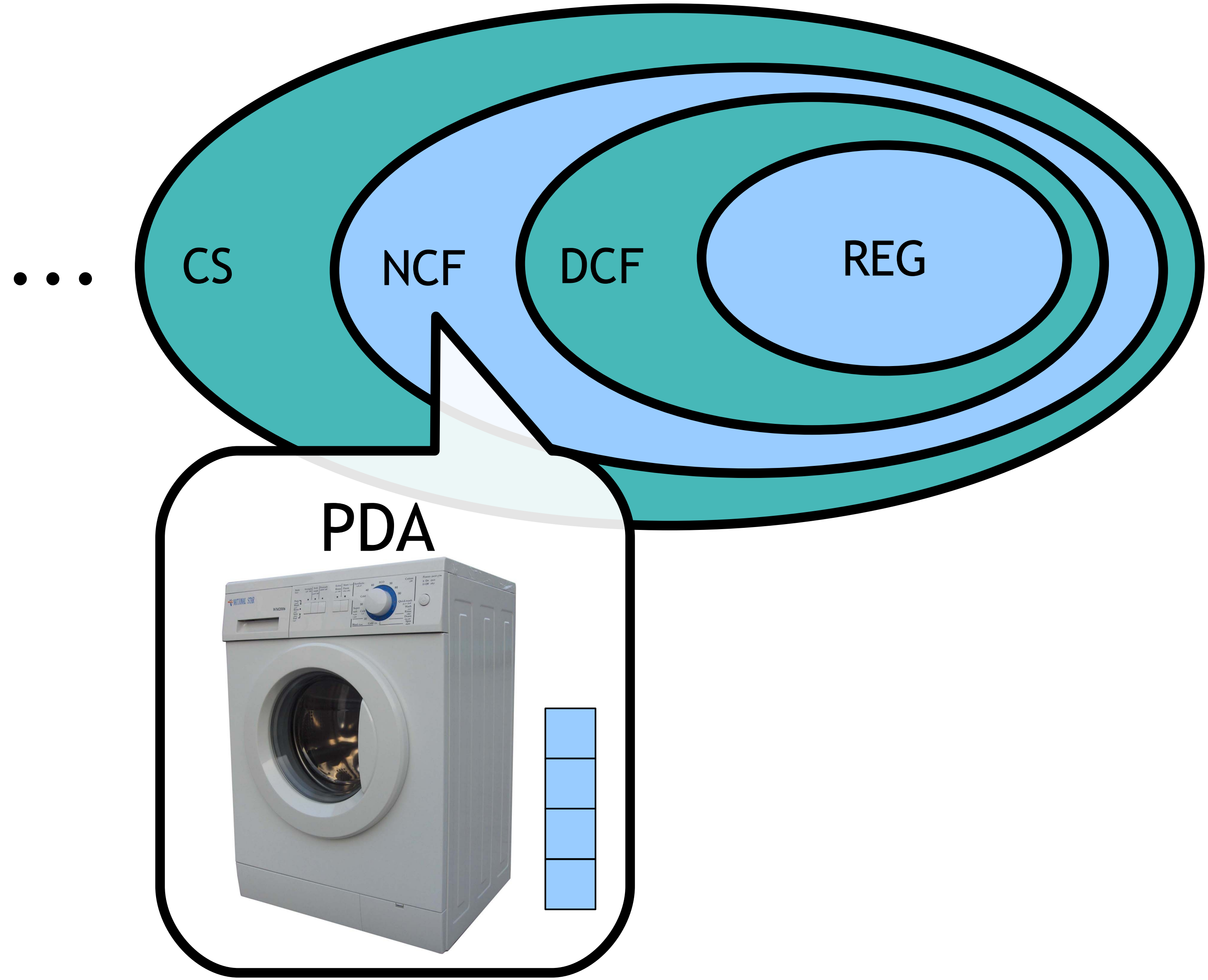


www.xkcd.com

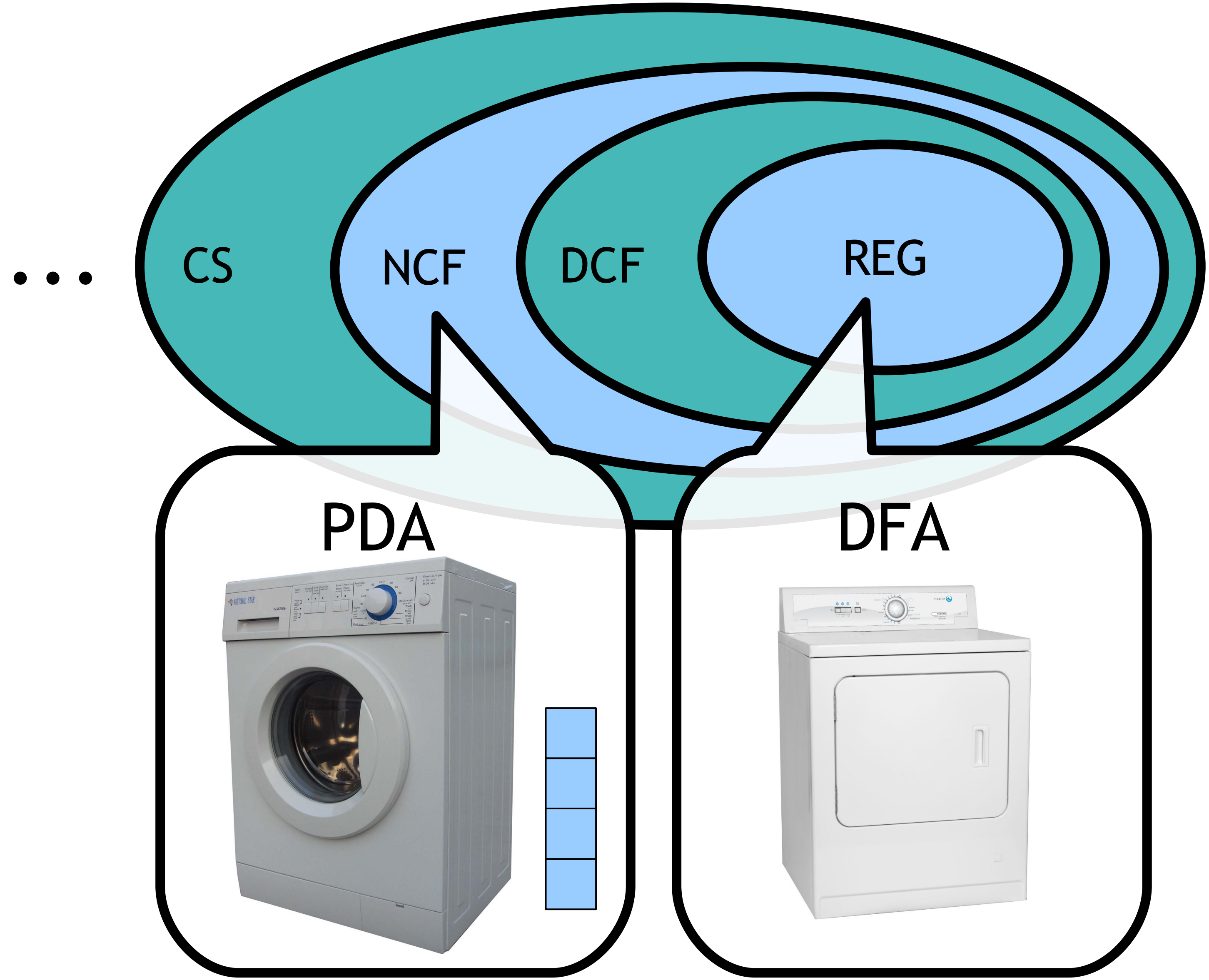
…Far Enough:



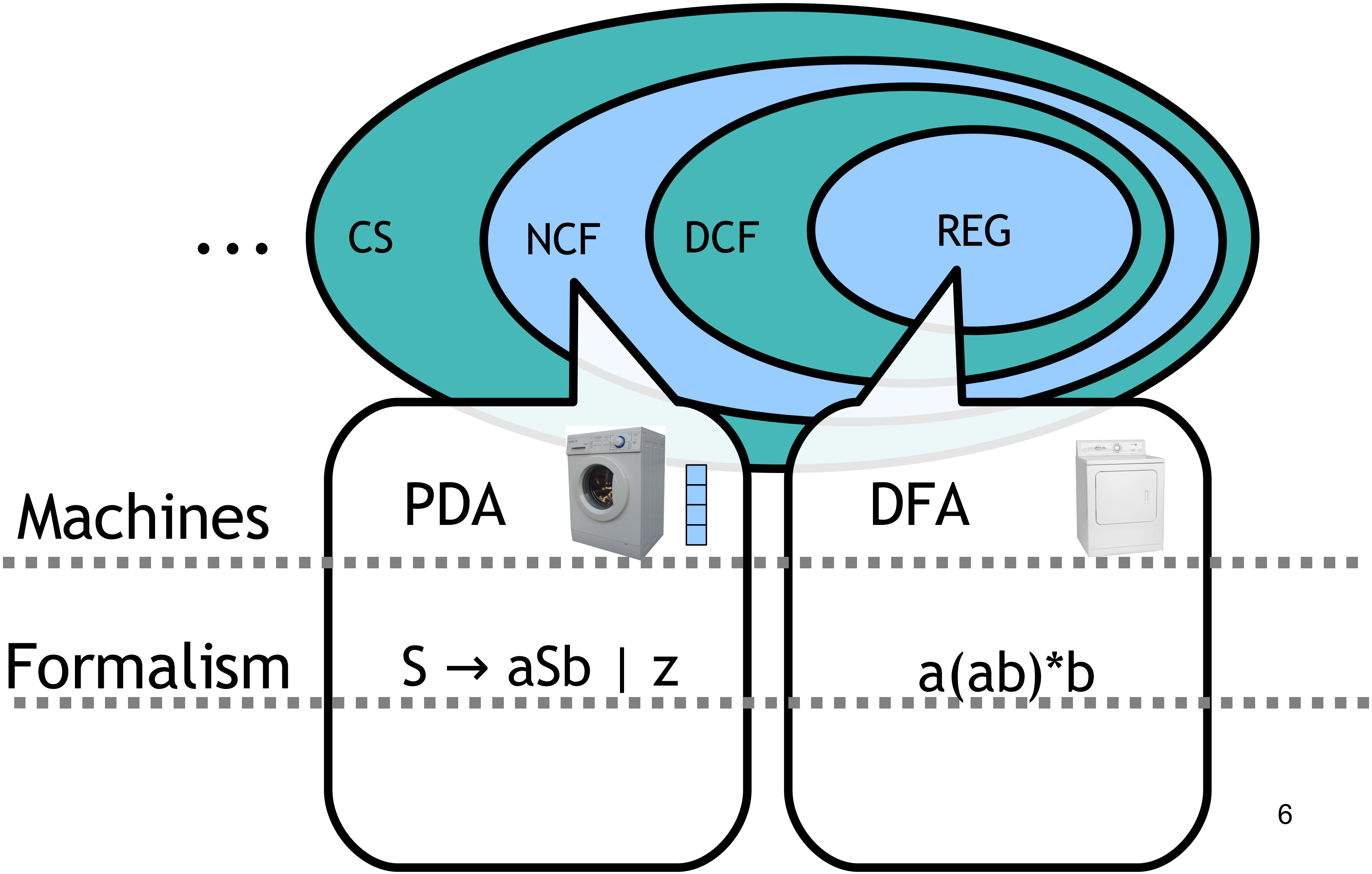
Formal Language Classes



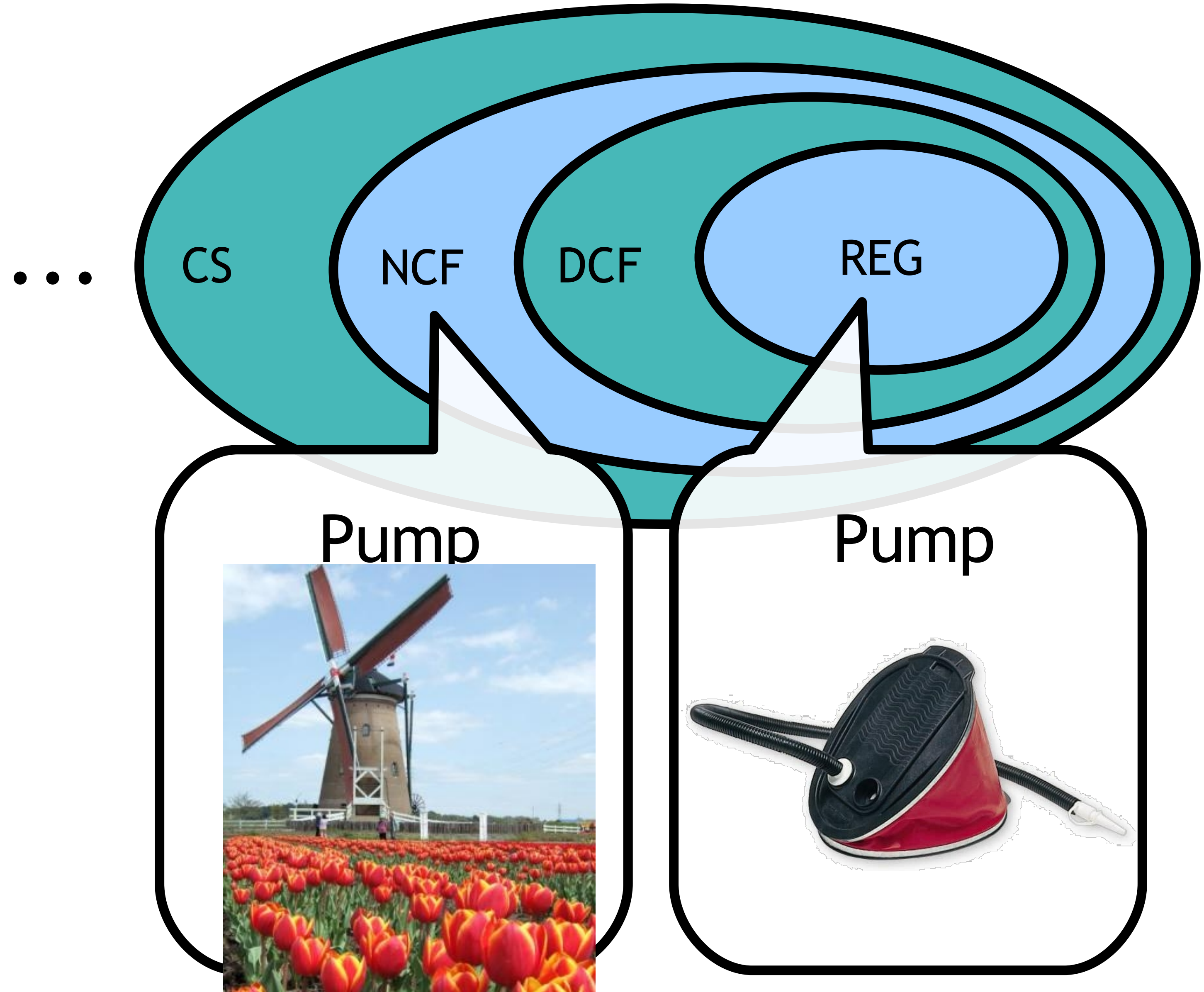
Formal Language Classes



Formal Language Classes

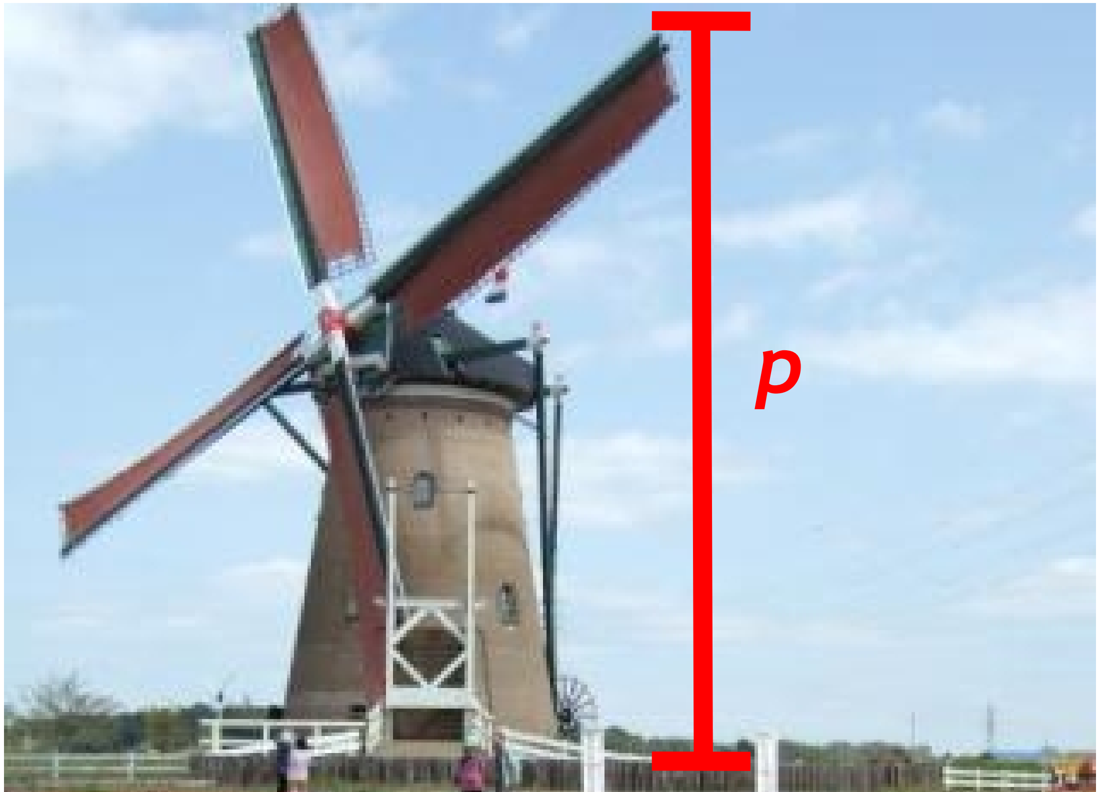


Formal Language Classes



The Context-Free Pumping Lemma

Suppose L = { $a^n b^n c^n$ | n = 1...} is context-free. By the pumping lemma, any string *s* with $|s| \geq p$ can be 'pumped.'



The Context-Free Pumping Lemma

What does 'can be pumped' mean?

$$s = uvxyz$$

1) $|vy| > 0$
2) $|vxy| \leq p$
3) uv^ixy^iz is in L for all $i \geq 0$

The Context-Free Pumping Lemma

Suppose L = { $a^nb^nc^n$ | n = 1...} is context-free.

Consider $s = a^pb^pc^p$; for any split $s = uvxyz$, we have:

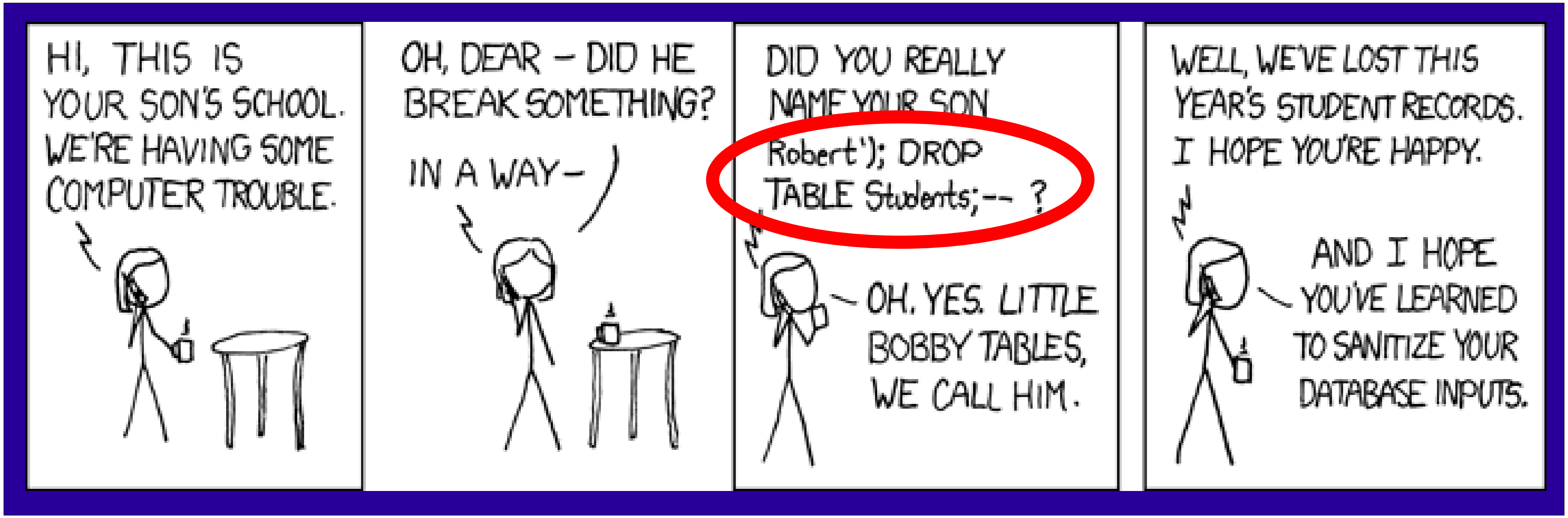
- if *v* contains *a's*, then *y* cannot contain *c*'s
- if *y* contains *c's*, then v cannot contain *a's*
- a problem: $uv^0xy^0z = uxz$ will never be in *L*

Moving Right Along

Motivation – Some Examples

- Can solve problems by phrasing them as 'language' problems:
  - Finding valid control flow graph paths
  - Solving set constraint problems
  - Static string analysis
  - Lexing, parsing
- For fun...

Example – String Variables



www.xkcd.com

The Nugget

Some Code:

```
x = 'z';

while(n < 5) {
  x = '(' . x . ')';
  n ++;
}
```

- We want a context free grammar to model **`x`**
- Suppose we don't know anything about **`n`**

The Nugget

Some Code:

```
>x = 'z';

 while(n < 5) {
   x = '(' . x . ')';
   n ++;
 }
```

Grammar:

```
A -> z
```

The Nugget

Some Code:

```
x = 'z';

while(n < 5) {
>  x = '(' . x . ')';
   n ++;
}
```

Grammar:

```
A -> z          [True]

B -> (A)        [n < 5]
```

The Nugget

Some Code:

```
 x = 'z';

 while(n < 5) {
   x = '(' . x . ')';
   n ++;
>>}
```

Grammar:

```
A -> z           [True]

B -> (A)         [n < 5]
C -> A | B       [n < 5]
```

The Nugget

Some Code:

```
 x = 'z';

 while(n < 5) {
   x = '(' . x . ')';
   n ++;
>>}
```

Grammar:

```
A -> z            [True]
B -> (C)         [n < 5]
C -> A | B        [True]
```

The Nugget

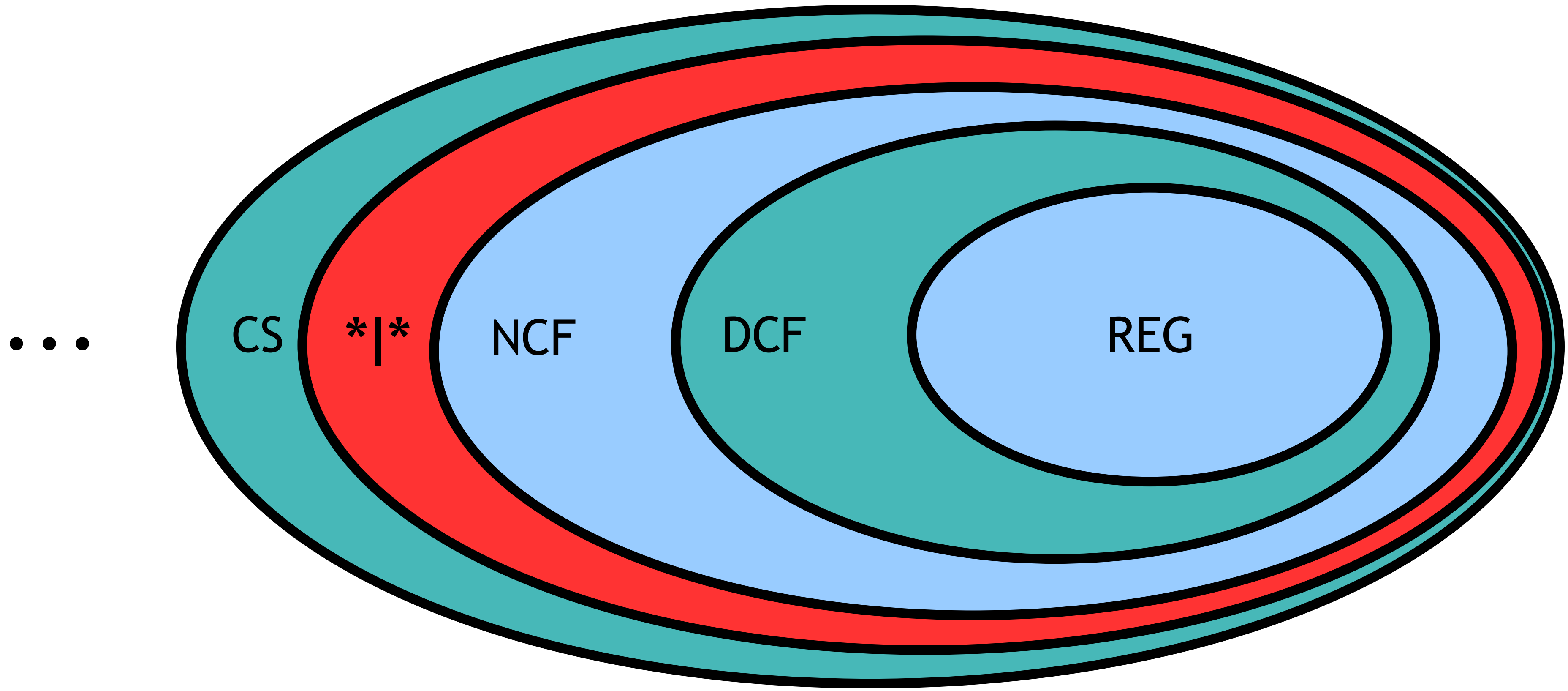
Some Code:

```
 x = 'z';

 while(n < 5) {
   x = '(' . x . ')';
   n ++;
>>}
```

Grammar:

```
X -> C

A -> z            [True]
B -> (C)        [n < 5]
C -> A | B        [True]
```

Indexed Languages



Definition: Indexed Grammar

- $G = (N, T, F, P, S)$

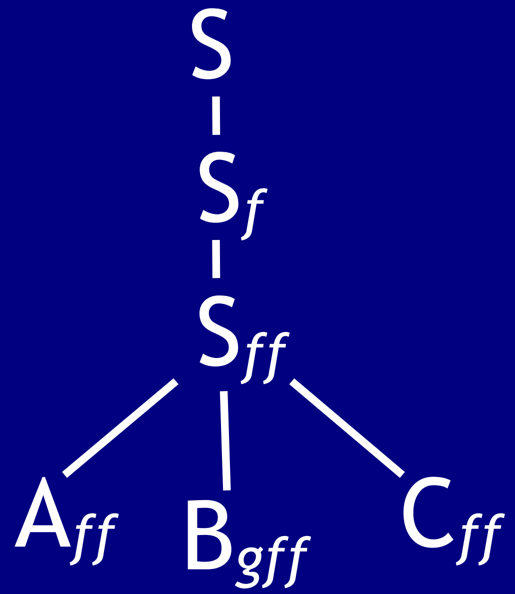
$$P \;:\; \{\, N \rightarrow ((NF^*) \cup T)^* \,\}$$
$$F \;:\; \{\, \{\, N \rightarrow (N \cup T)^* \}_f \,\}$$

Derivation Rule Example

1

$S \rightarrow S_f$

$S \rightarrow AB_gC$



Derivation Rule Example

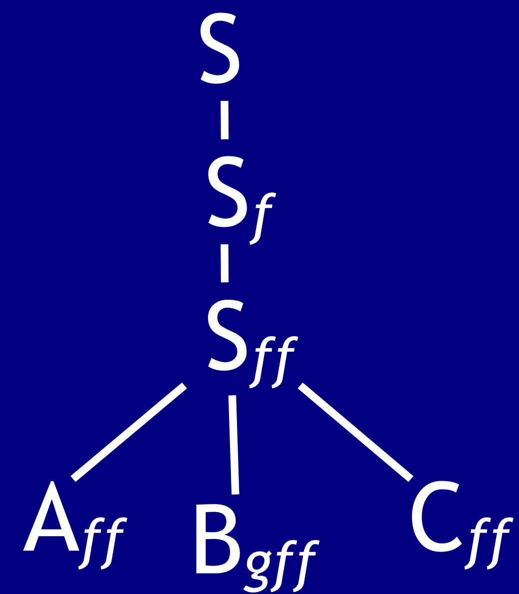
①

$S \rightarrow S_f$

$S \rightarrow AB_gC$

'normal' production:

copy index stack

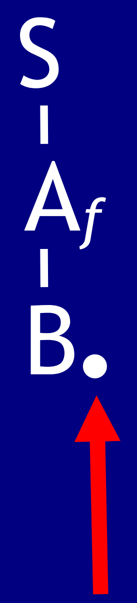


Derivation Rule Example

2

$S \longrightarrow A_f$

$F = \{ \ \{ A \longrightarrow B \}_f \ \}$



Derivation Rule Example

2

$S \rightarrow A_f$

$F = \{ \ \{ A \rightarrow B\}_f \ \}$

S

$|$

A_f

$|$

B

Derivation Rule Example

2

$S \rightarrow A_f$

$F = \{ \; \{ A \rightarrow B \}_f \; \}$

'index' production:

pop leftmost index

(for current nonterminal only)

S

|

A_f

|

B

An Example Grammar

$S \to D_f$

$D \to D_g \mid ABC$

$g = \{A \to Aa$
$\quad B \to Bb$
$\quad C \to Cc\}$

$f = \{A \to a$
$\quad B \to b$
$\quad C \to c\}$

An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

$\quad B \rightarrow Bb$

$\quad C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$\quad B \rightarrow b$

$\quad C \rightarrow c\}$

An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

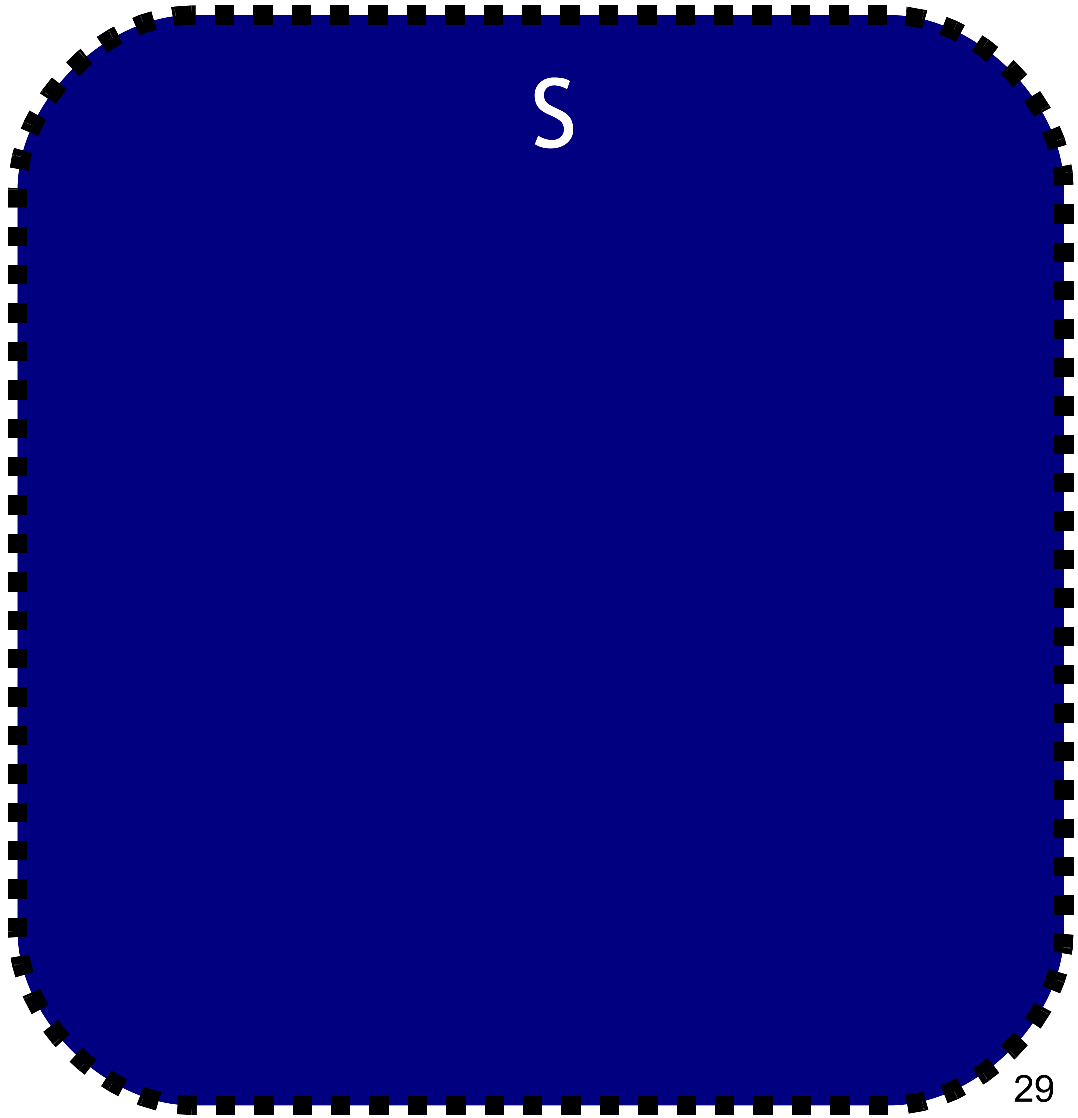
$B \rightarrow Bb$

$C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c\}$



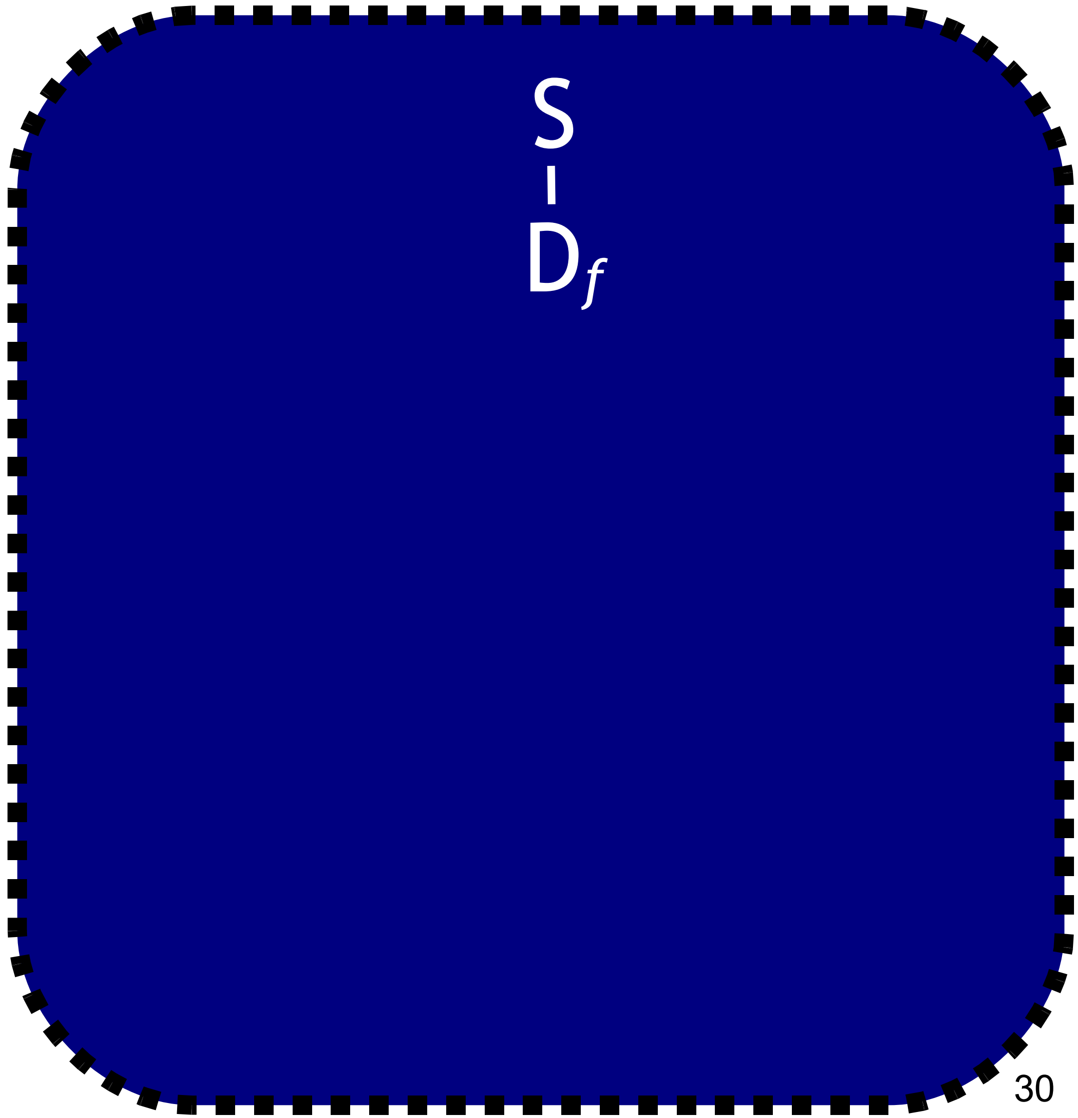
An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$
$\quad B \rightarrow Bb$
$\quad C \rightarrow Cc\}$

$f = \{A \rightarrow a$
$\quad B \rightarrow b$
$\quad C \rightarrow c\}$



An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$
$\quad B \rightarrow Bb$
$\quad C \rightarrow Cc\}$

$f = \{A \rightarrow a$
$\quad B \rightarrow b$
$\quad C \rightarrow c\}$

An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

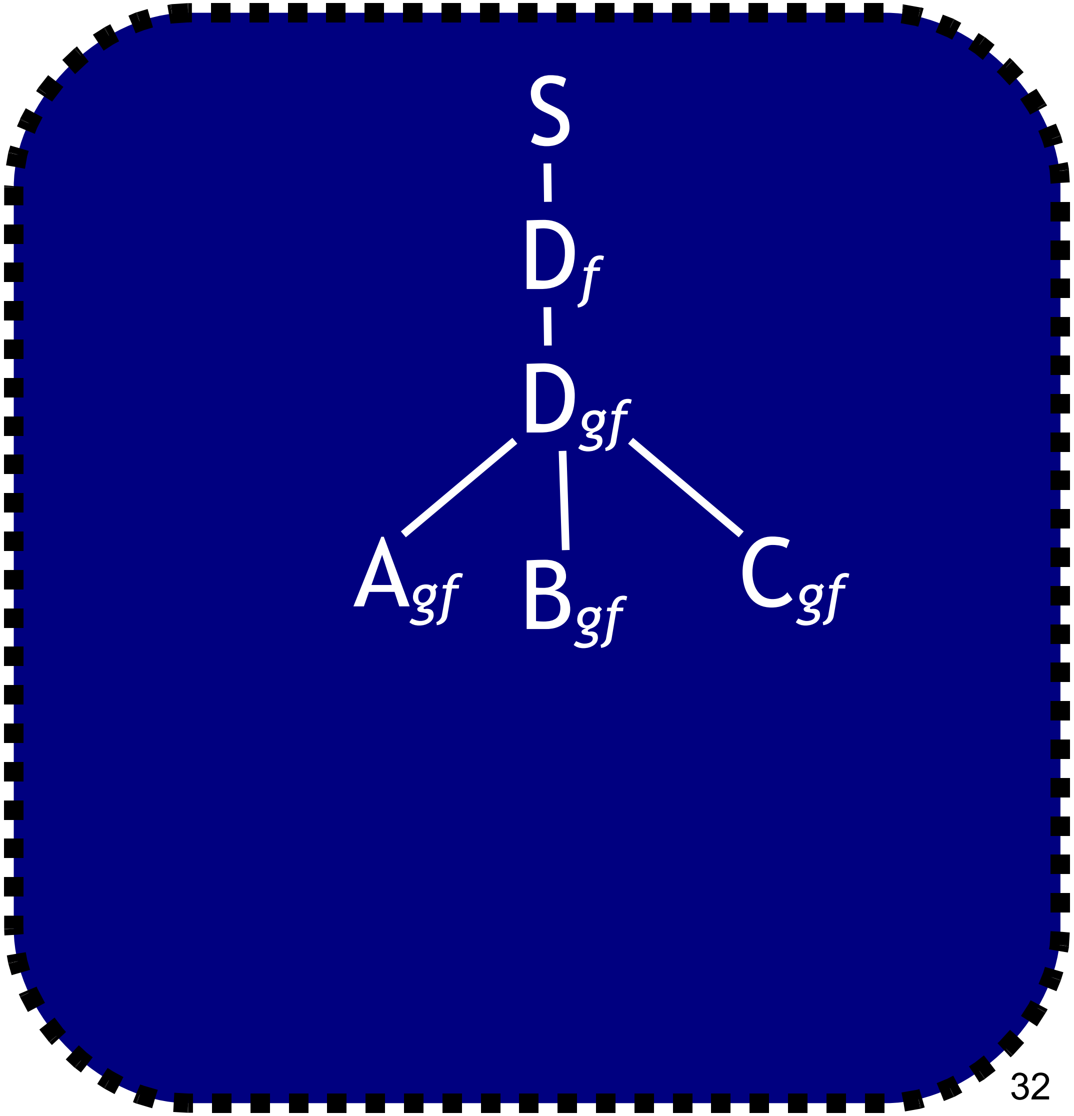
$B \rightarrow Bb$

$C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c\}$



An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

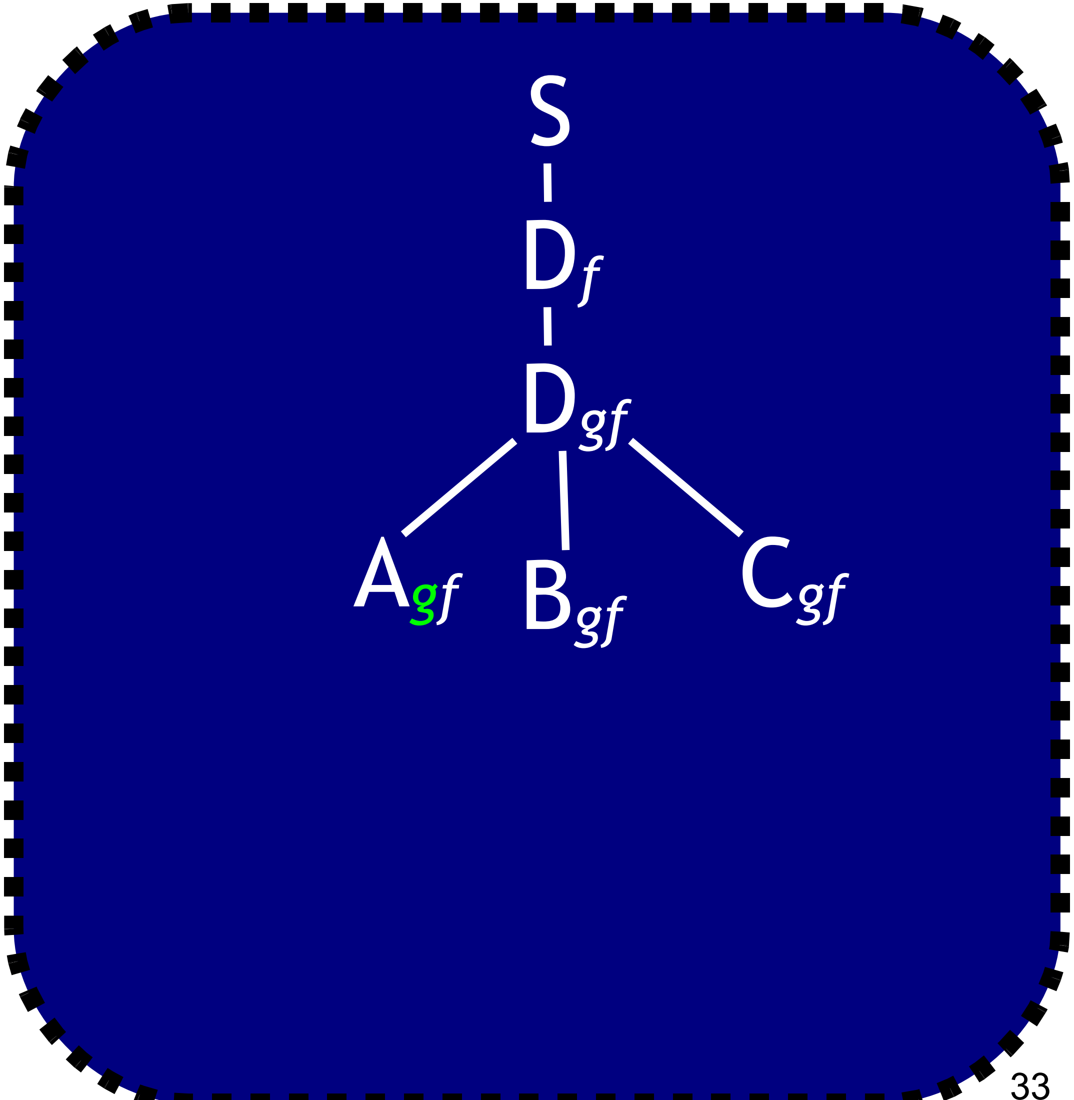
$B \rightarrow Bb$

$C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c\}$



An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

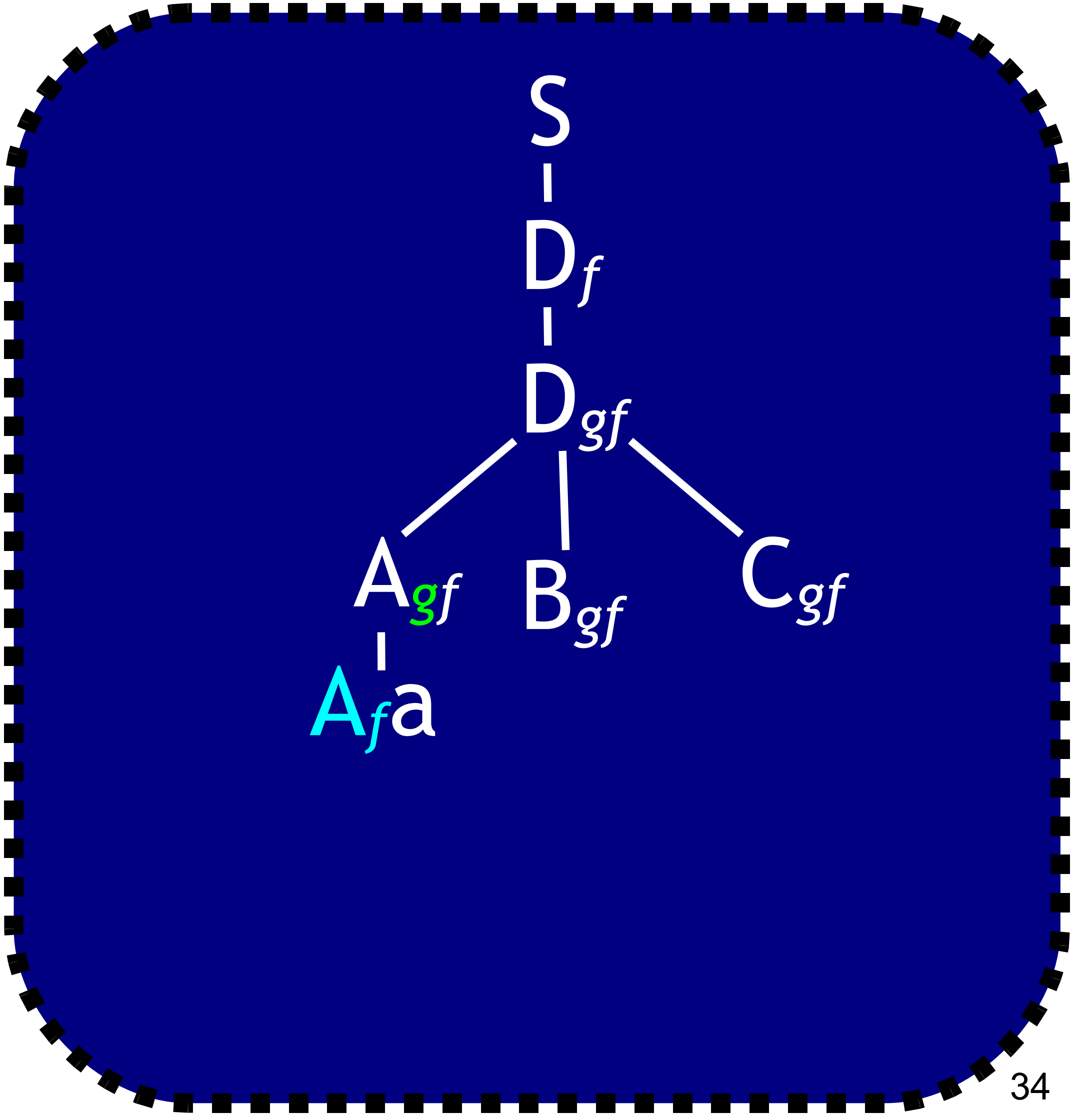
$B \rightarrow Bb$

$C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c\}$



An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

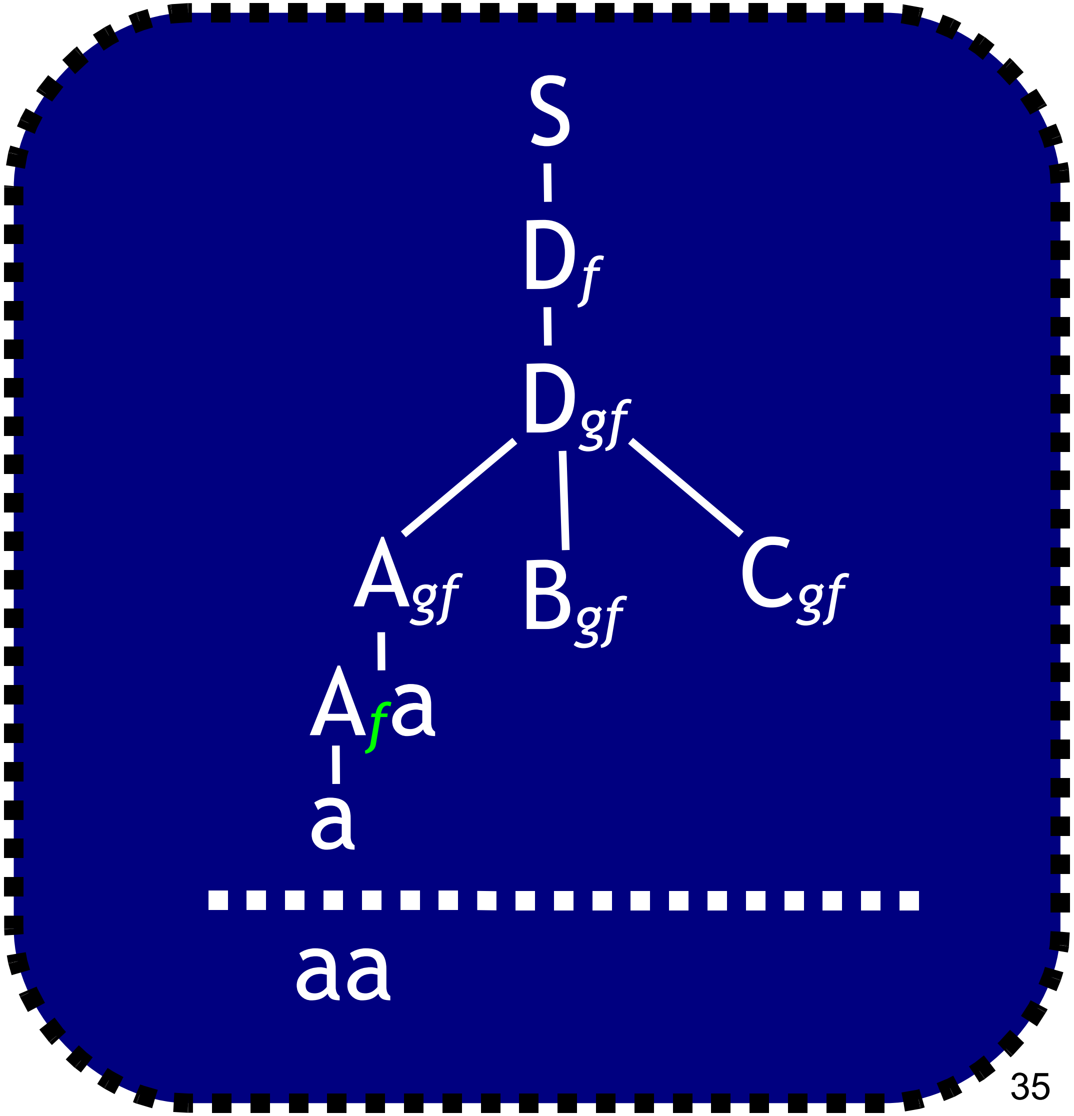
$B \rightarrow Bb$

$C \rightarrow Cc\}$

$f = \{A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c\}$



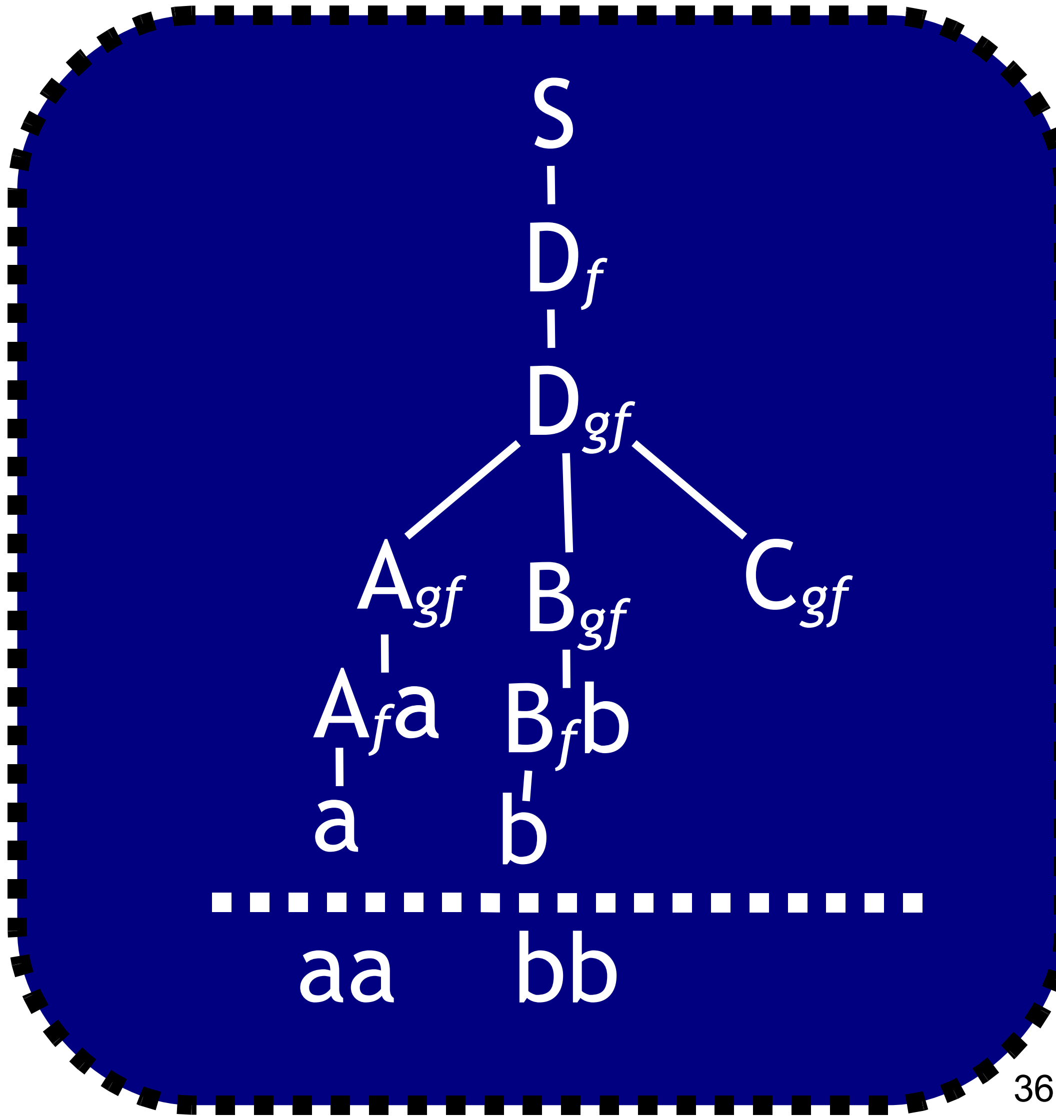
An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$$g = \{A \rightarrow Aa$$
$$B \rightarrow Bb$$
$$C \rightarrow Cc\}$$

$$f = \{A \rightarrow a$$
$$B \rightarrow b$$
$$C \rightarrow c\}$$



An Example Grammar

$S \rightarrow D_f$

$D \rightarrow D_g \mid ABC$

$g = \{A \rightarrow Aa$

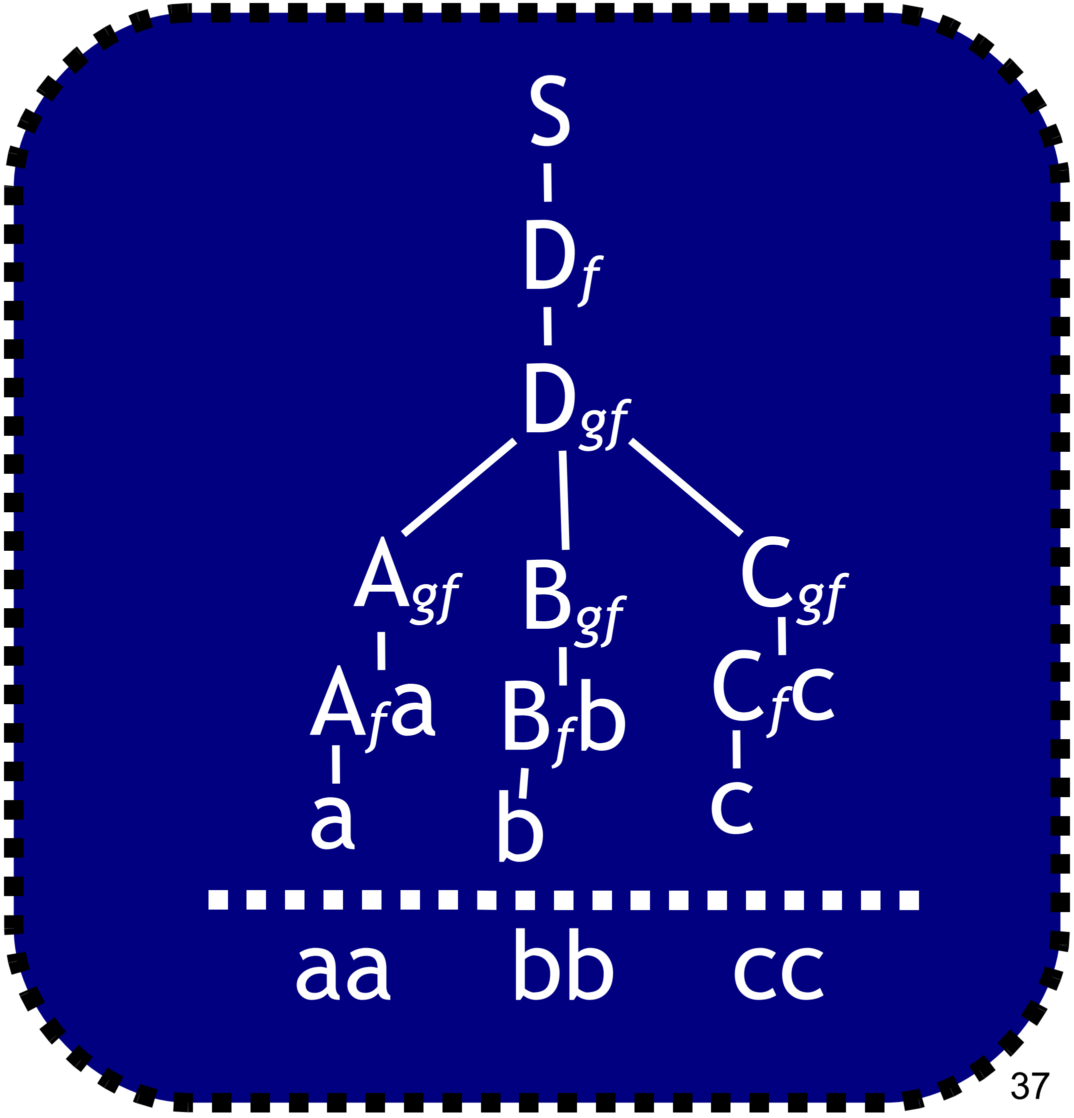
$\quad B \rightarrow Bb$

$\quad C \rightarrow Cc\}$

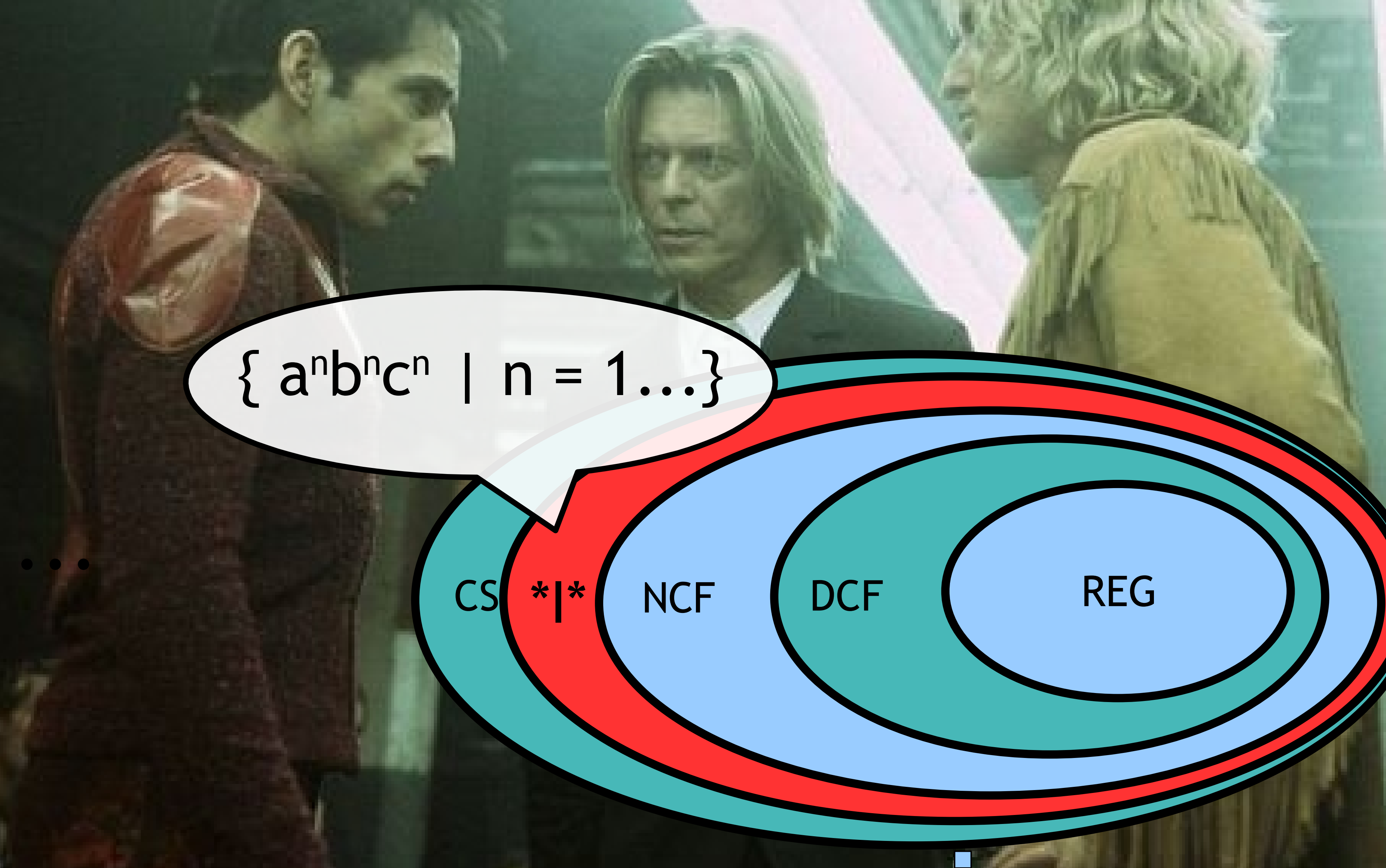
$f = \{A \rightarrow a$

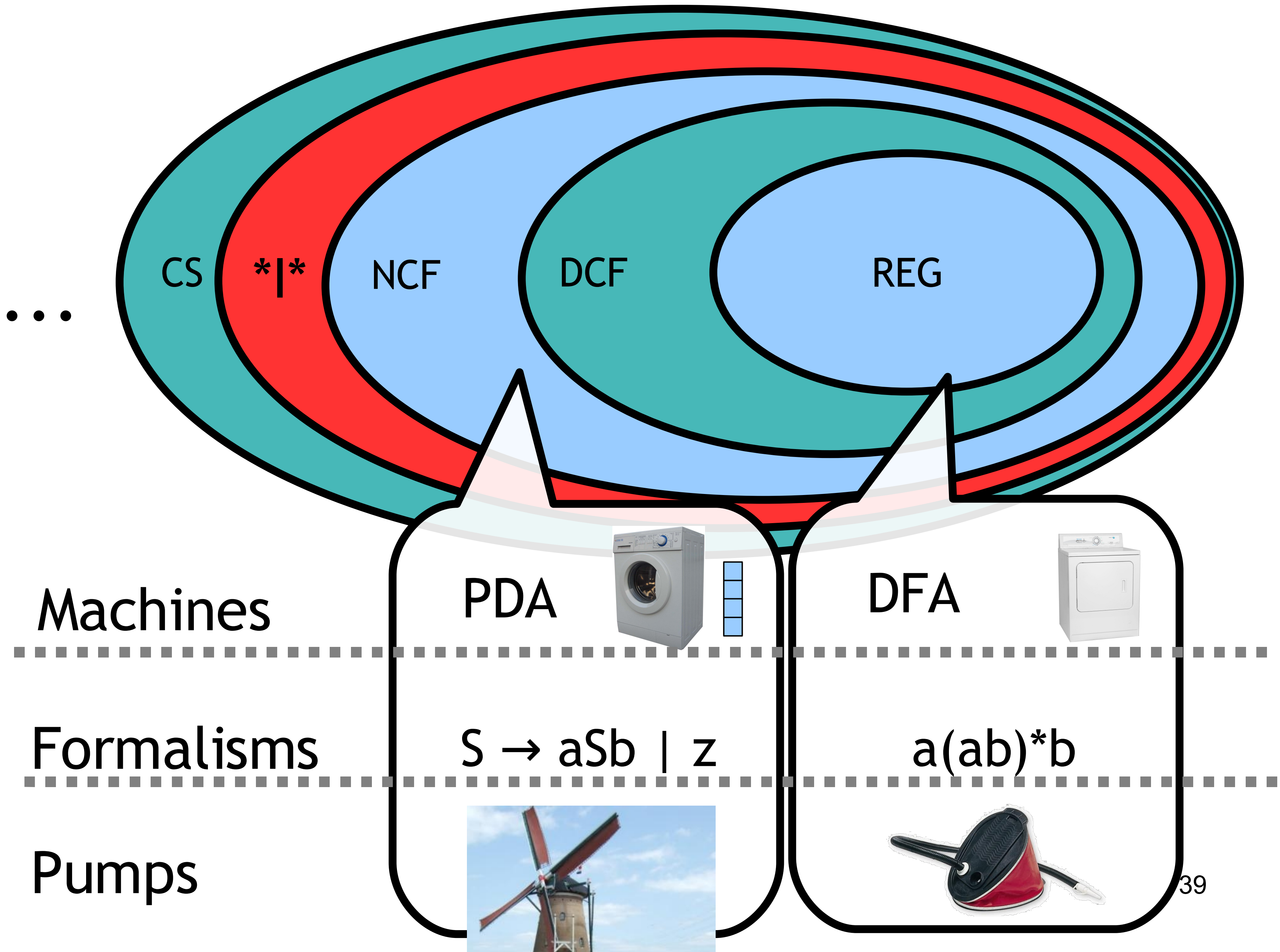
$\quad B \rightarrow b$

$\quad C \rightarrow c\}$

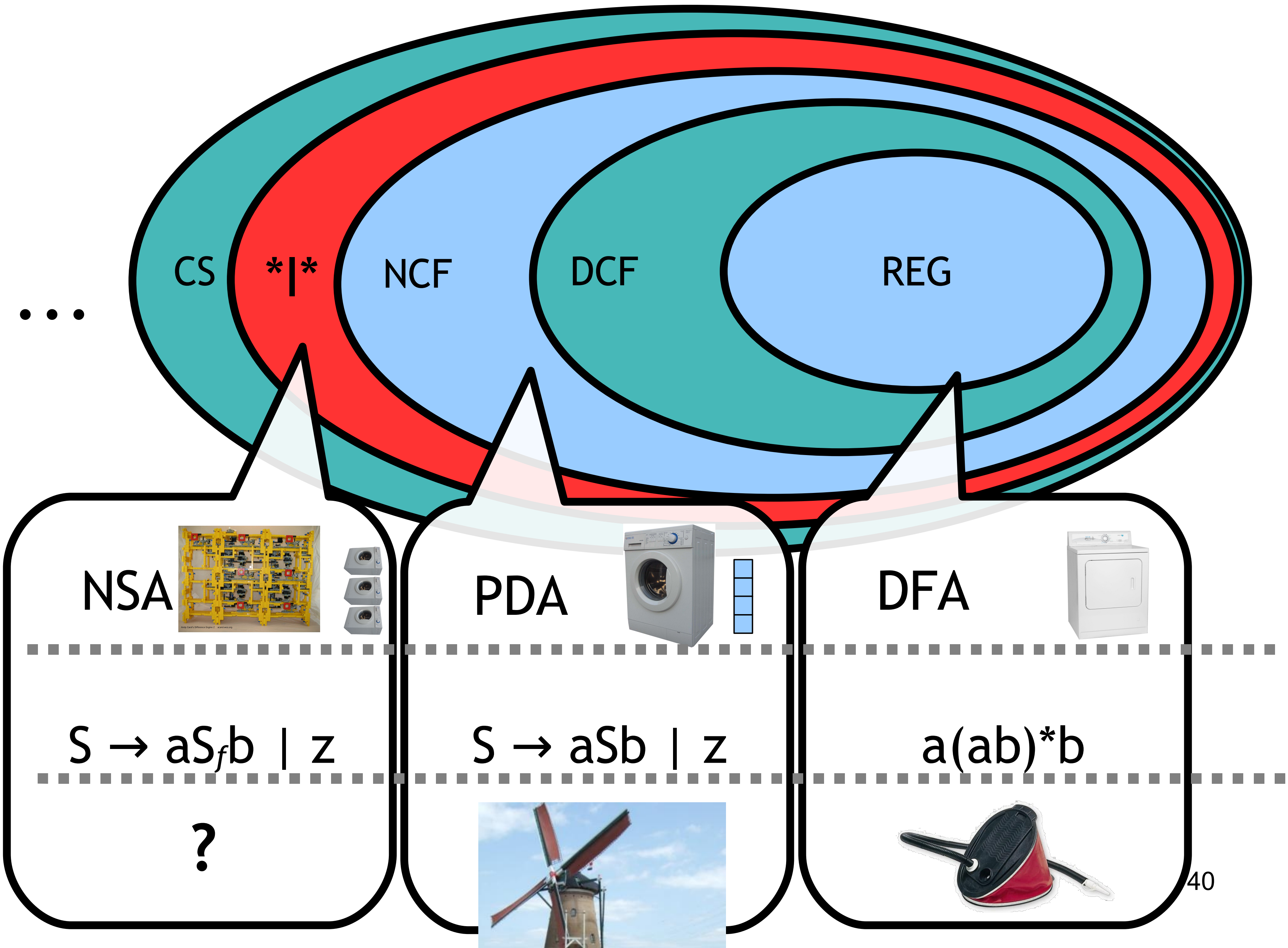


Where am I?

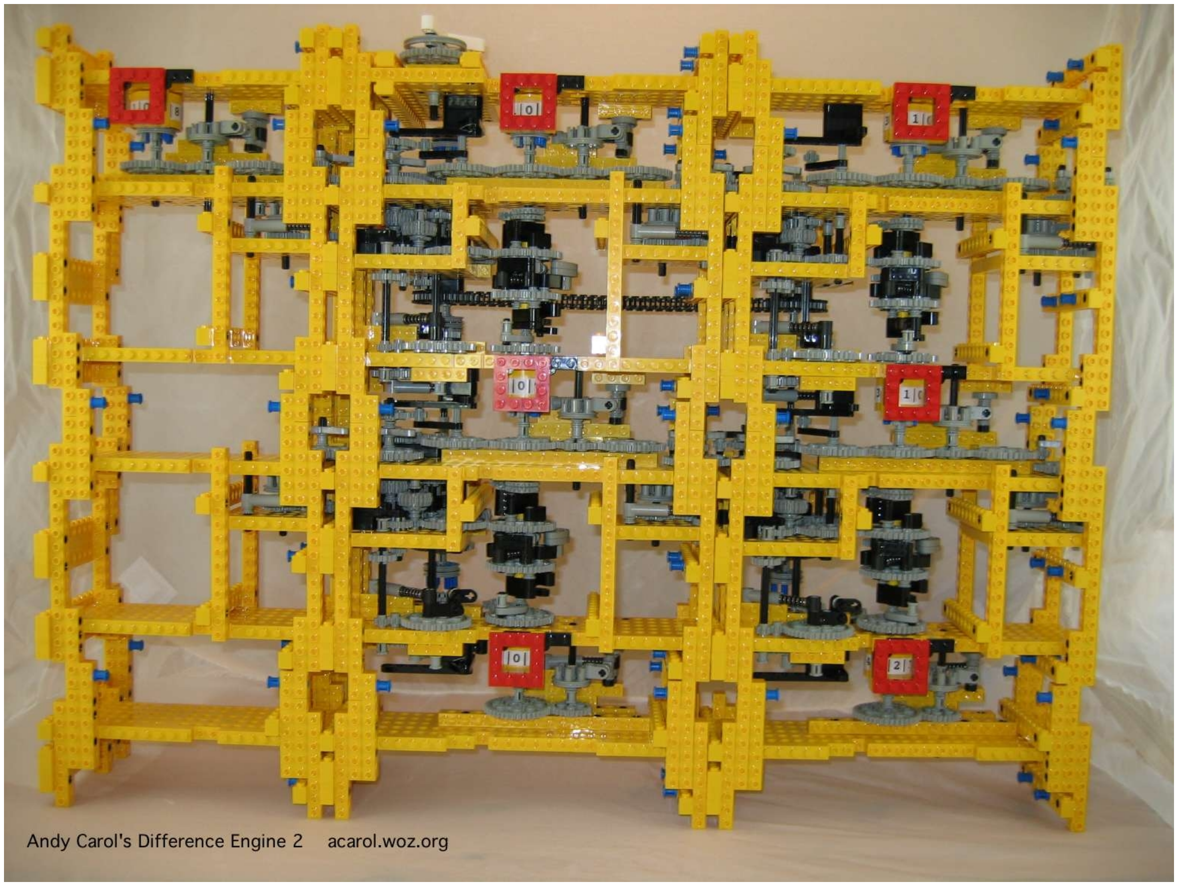




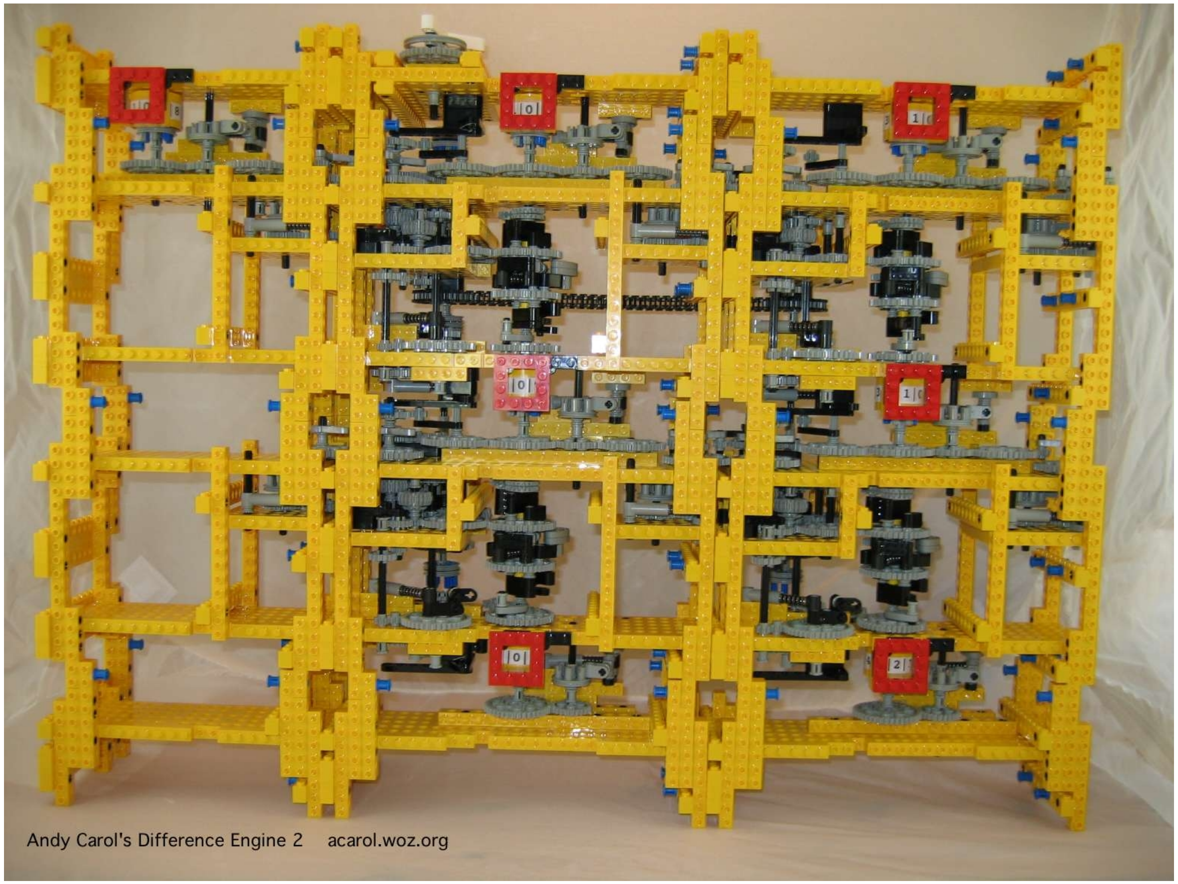
...
CS
|
NCF
DCF
REG
Machines
PDA
DFA
Formalisms
S → aSb | z
a(ab)*b
Pumps


...
CS
I
NCF
DCF
REG
NSA
$S \to aS_f b \mid z$
?
PDA
$S \to aSb \mid z$
DFA
a(ab)*b

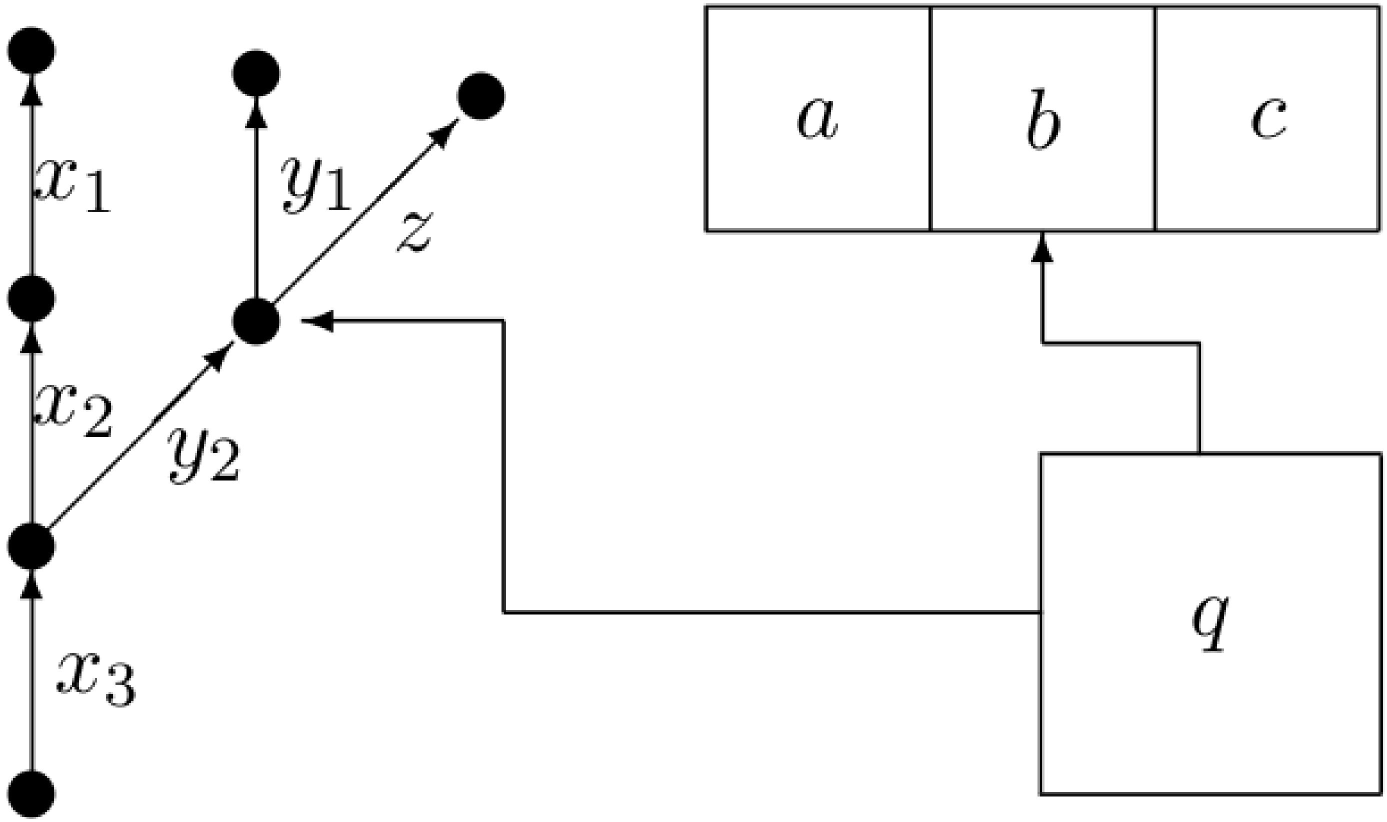
Associated Automaton



Associated Automaton



(Actual NSA)



[Gilman, Shapiro 1998]

But will it pump?

- It's complicated – on derivation trees rather than strings [Hayashi 1973]

- Several corollaries exist [Gilman 1996] (less general; easier to use)

The Lemma

L : indexed language m : positive integer

There is a constant $k > 0$ so that each w in L can be split ($w = w_1 w_2 \ldots w_r$) subject to:

The Lemma

L : indexed language m : positive integer

There is a constant $k > 0$ so that each w in L can be split ($w = w_1 w_2 \ldots w_r$) subject to:

- $m < r \leq k$
- each $|w_i| > 0$

The Lemma

L : indexed language m : positive integer

There is a constant $k > 0$ so that each w in L can be split ($w = w_1 w_2 \ldots w_r$) subject to:

- $m < r \leq k$
- each $|w_i| > 0$
- any m-sized set of w_i's is a subset of some w' in L; w' is a subproduct of w

Lemma Example

Suppose $L = \{(ab^n)^n \ \mid n \in \mathbb{N}\}$ is indexed. Let $m = 1$.

Consider $w = (ab^n)^n$ with $n > k$, which can be split into r subproducts: $w = w_1 w_2 \ldots w_r$. Since $r \leq k$, at least one w_i must contain two or more a's.

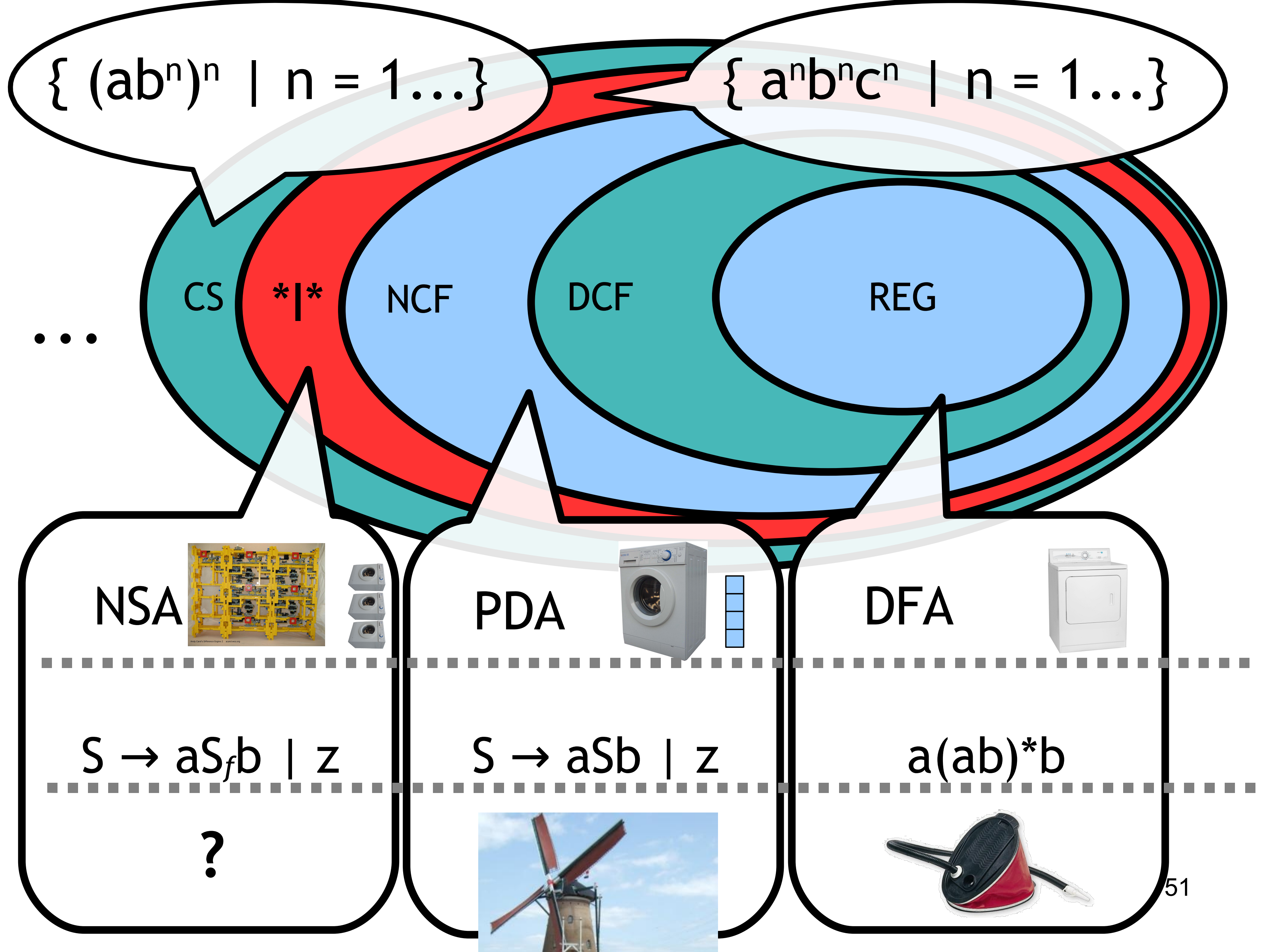
Lemma Example

Suppose $L = \{(ab^n)^n \mid n \in \mathbb{N}\}$ is indexed. Let $m = 1$.

Consider $w = (ab^n)^n$ with $n > k$, which can be split into r subproducts: $w = w_1 w_2 \ldots w_r$. Since $r \leq k$, at least one w_i must contain two or more a's.

Pick such a w_i. Any w' that contains w_i must contain the substring $ab^n a$. Contradiction: w' cannot simultaneously be a substring of w and contain $ab^n a$.


(Montage Time)


$\{ (ab^n)^n \mid n = 1 \ldots \}$
$\{ a^n b^n c^n \mid n = 1 \ldots \}$
CS
|
NCF
DCF
REG
...
NSA
PDA
DFA
S → aS$_f$b | z
S → aSb | z
a(ab)*b
?

References

- Indexed Grammars – An Extension to Context-Free Grammars (Aho)
- Nested Stack Automata (Aho)
- Sequentially Indexed Grammars (van Eijck)
- A Shrinking Lemma for Indexed Languages (Gilman)
- On Groups Whose Word Problem is Solved by a Nested Stack Automaton (Gilman and Shapiro)
- On Derivation Trees of Indexed Grammars (Hayashi)