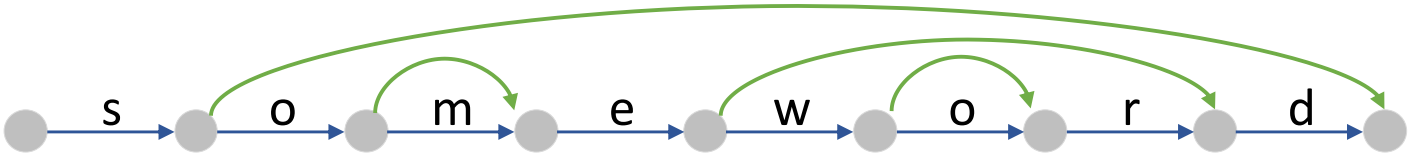
Colored Nested Words

Rajeev Alur and Dana Fisman

LATA'16
Prague, Czech Republic

Nested Words

- A model of linear and hierarchical structure in data.



- Examples of such data
  - Executions of procedural programs
  - Marked-up languages such as XML
  - Annotated linguistic data

Example: Execution of a Structured Program as a Nested Word

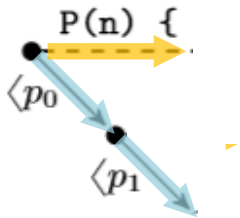
3 procedures

```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```

Example: Execution of a Structured Program as a Nested Word

3 procedures

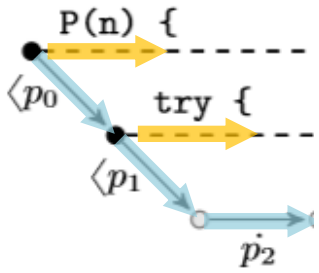
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

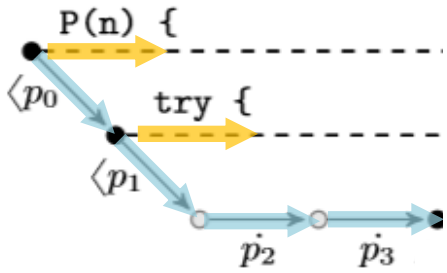
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

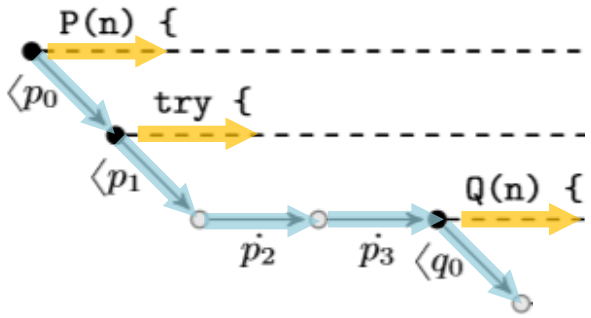
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

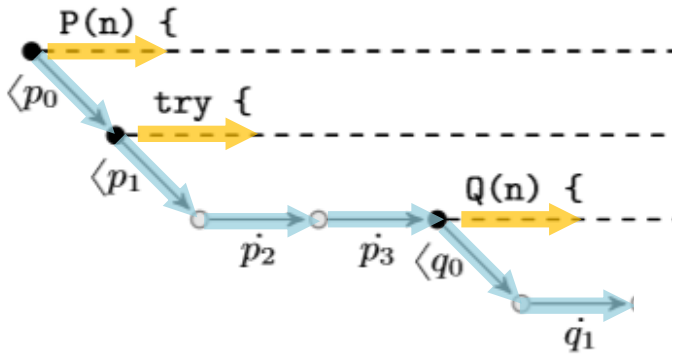
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

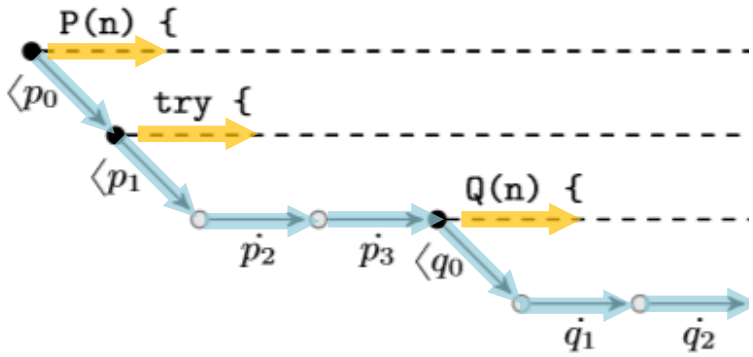
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



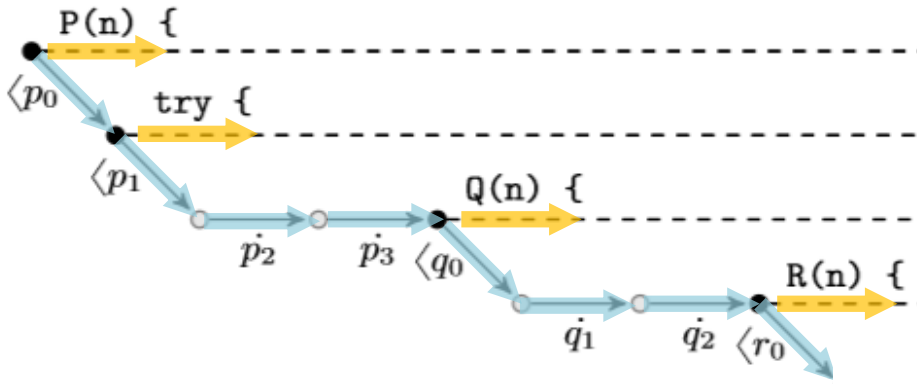
Example: Execution of a Structured Program as a Nested Word

3 procedures

```
p0: P(n) {
p1:     try {
p2:         x = 2*n
p3:         x = Q(x)
p4:     }
p5:     catch (int) {}
p6:     return x }
```

```
q0: Q(n) {
q1:     y = n / 2
q2:     y = R(y)
q3:     y = y+1
q4:     return y }
```

```
r0: R(n) {
r1:     z = n-1
r2:     if (z<0)
r3:         throw 0
r4:     return z }
```



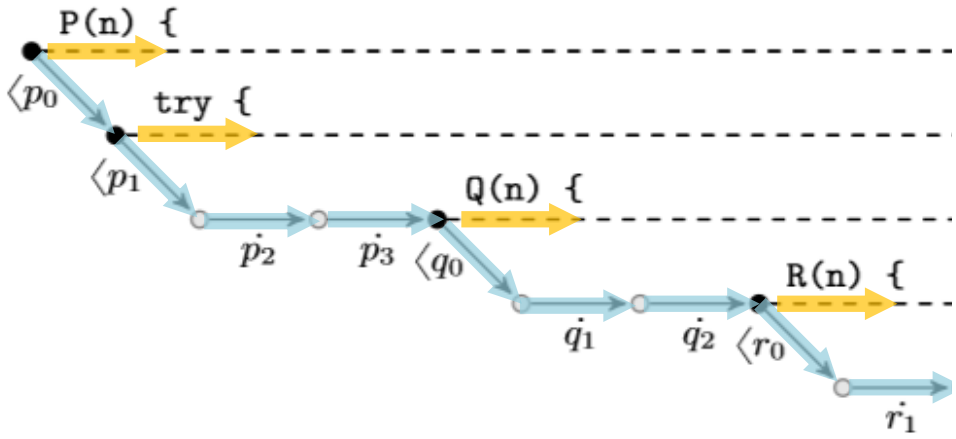
Example: Execution of a Structured Program as a Nested Word

3 procedures

```
p0: P(n) {
p1:     try {
p2:         x = 2*n
p3:         x = Q(x)
p4:     }
p5:     catch (int) {}
p6:     return x }
```

```
q0: Q(n) {
q1:     y = n / 2
q2:     y = R(y)
q3:     y = y+1
q4:     return y }
```

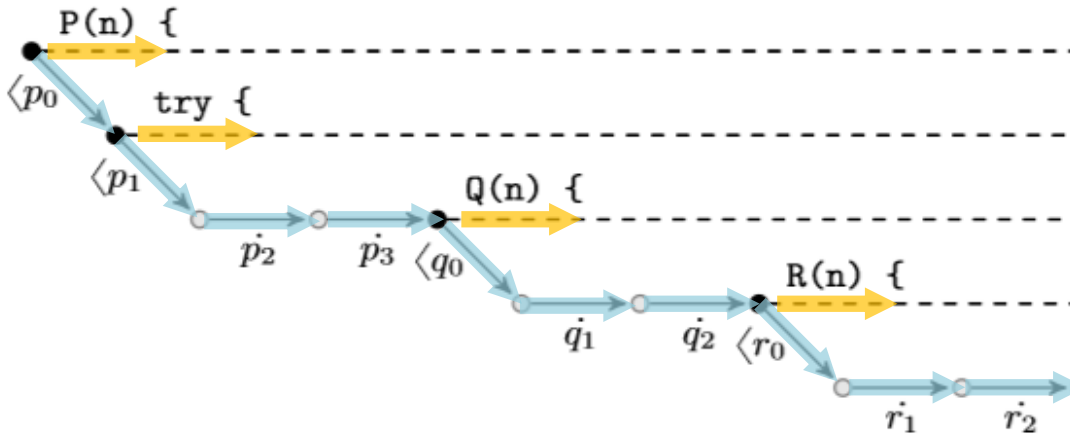
```
r0: R(n) {
r1:     z = n-1
r2:     if (z<0)
r3:         throw 0
r4:     return z }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

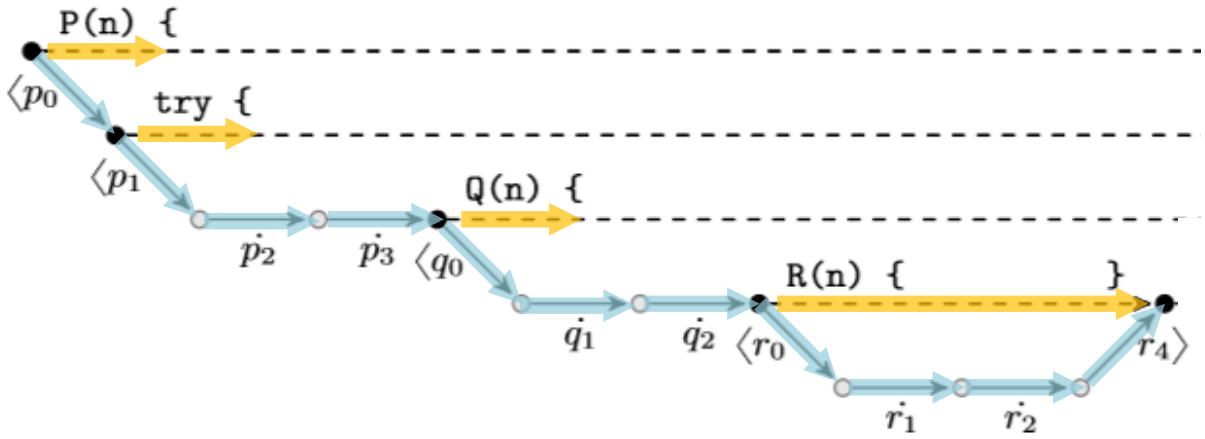
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

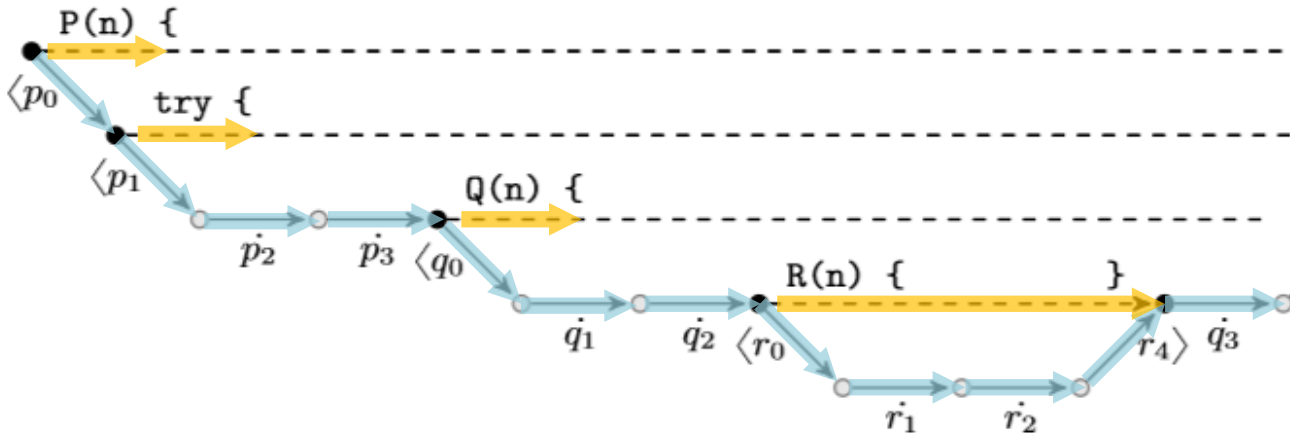
```
p0: P(n) {                q0: Q(n) {                r0: R(n) {
p1:     try {             q1:     y = n / 2         r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)          r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1           r3:         throw 0
p4:     }                 q4:     return y }        r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

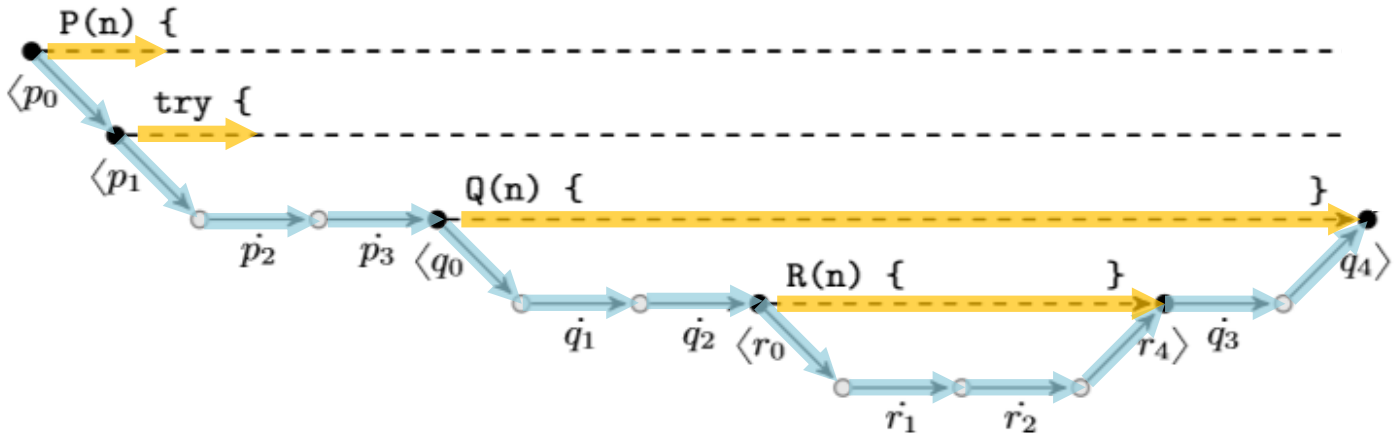
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

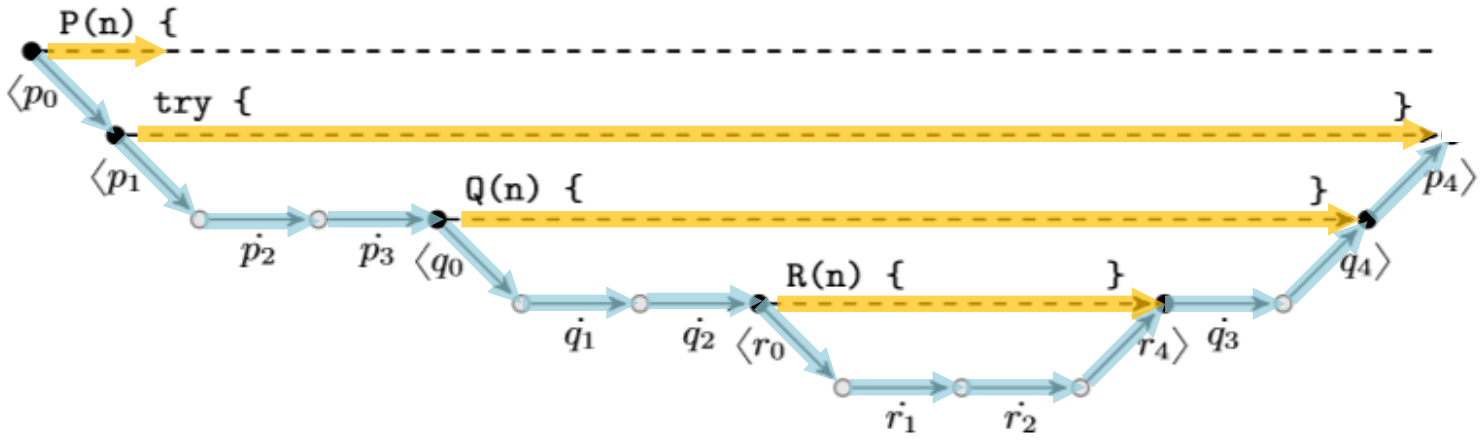
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Example: Execution of a Structured Program as a Nested Word

3 procedures

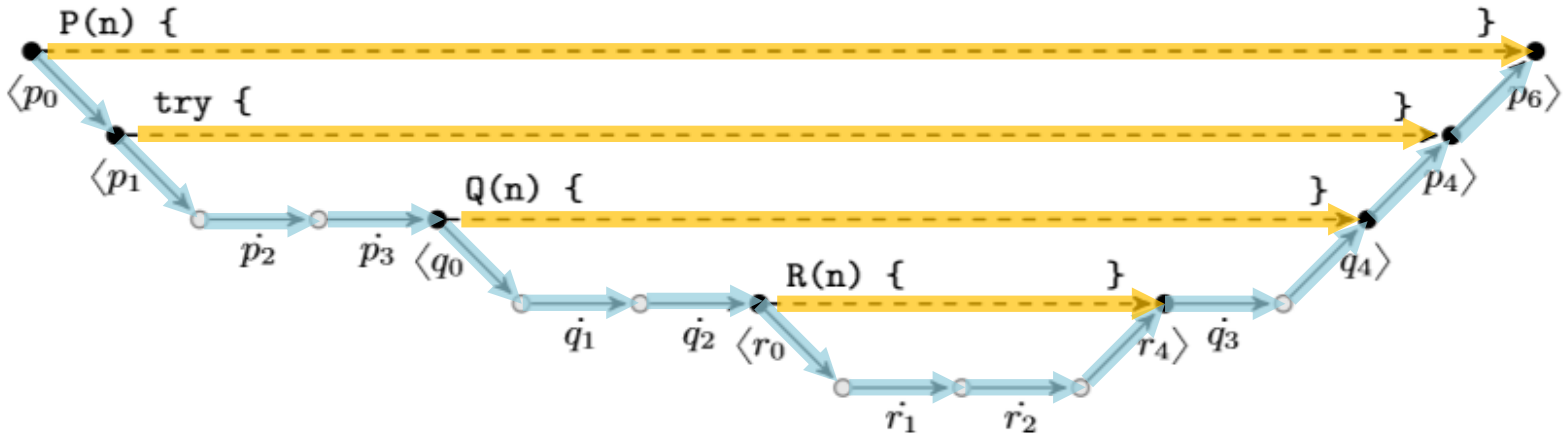
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



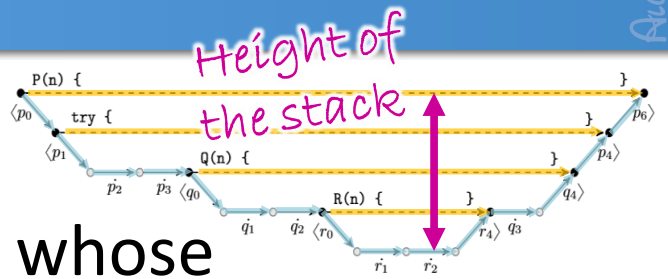
Example: Execution of a Structured Program as a Nested Word

3 procedures

```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Nested Words Automata (NWA)



- Special kinds of pushdown automata whose push/pop actions are directed by the hierarchical structure in the input nested word.
  - On every open-tag, the automaton pushes
  - On every close-tag, the automaton pops
  - On data with no hierarchical connections, no stack operation is made.
- A language of nested words is said to be regular if it can be recognized by a NWA

Properties of Regular Languages of Nested Words Automata

- Deterministic nested word automata are as expressive as their non-deterministic counterparts

- The class is closed under:
  - union
  - intersection
  - complementation
  - concatenation
  - Kleene-*
  - prefixes
  - reversal
  - homomorphism

- The following problems are decidable:
  - emptiness
  - membership
  - inclusion
  - equivalence

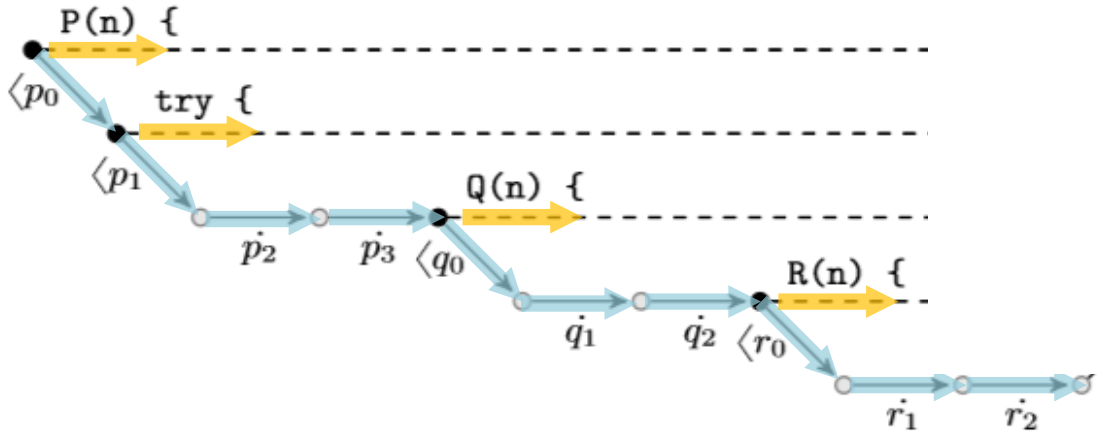
Weakness of Nested Words

- But what if an exceptions is thrown?

```
p0: P(n) {
p1:     try {
p2:         x = 2*n
p3:         x = Q(x)
p4:     }
p5:     catch (int) {}
p6:     return x }
```

```
q0: Q(n) {
q1:     y = n / 2
q2:     y = R(y)
q3:     y = y+1
q4:     return y }
```

```
r0: R(n) {
r1:     z = n-1
r2:     if (z<0)
r3:         throw 0
r4:     return z }
```



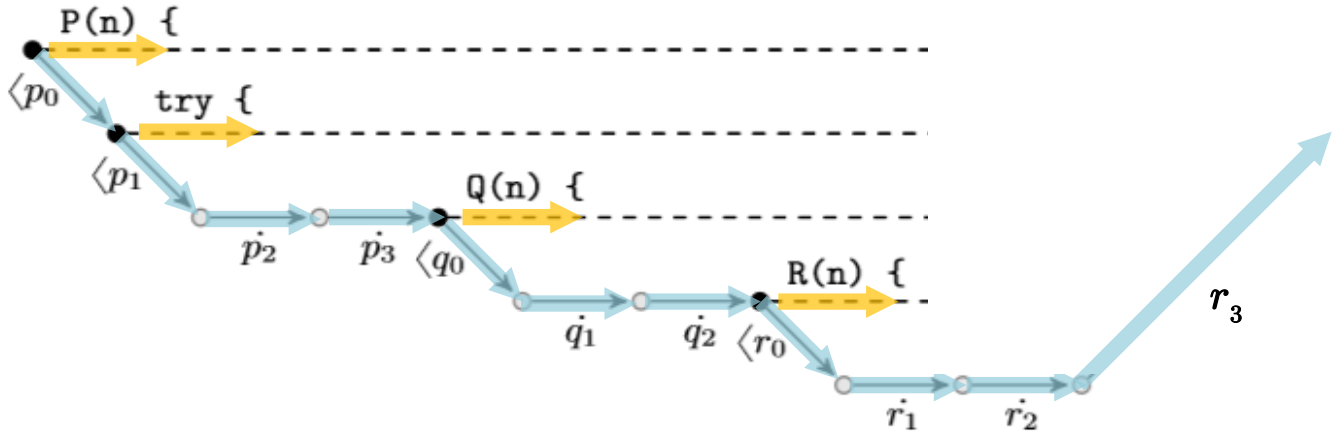
Weakness of Nested Words

- But what if an exceptions is thrown?

```
p0: P(n) {
p1:     try {
p2:         x = 2*n
p3:         x = Q(x)
p4:     }
p5:     catch (int) {}
p6:     return x }
```

```
q0: Q(n) {
q1:     y = n / 2
q2:     y = R(y)
q3:     y = y+1
q4:     return y }
```

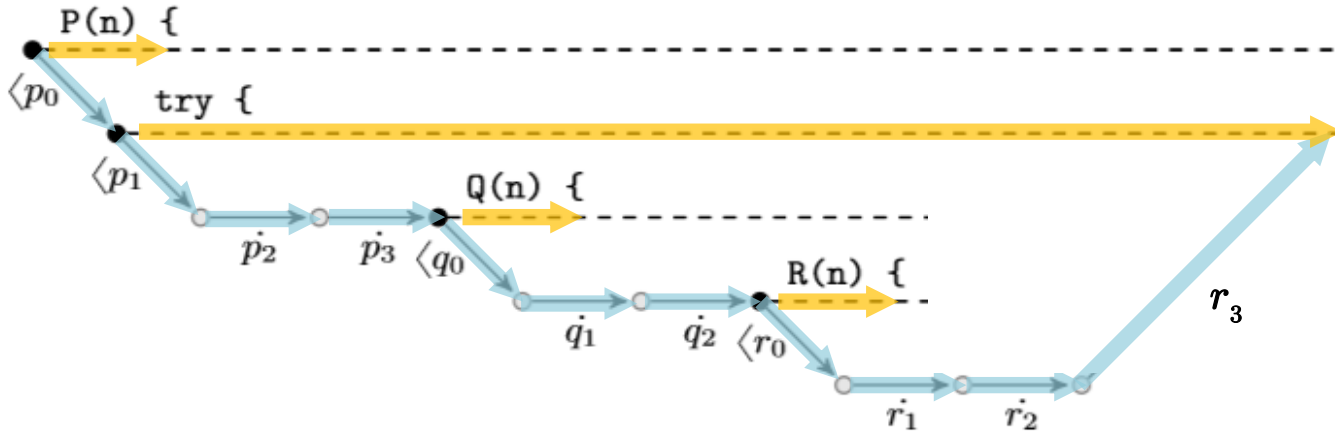
```
r0: R(n) {
r1:     z = n-1
r2:     if (z<0)
r3:         throw 0
r4:     return z }
```



Weakness of Nested Words

- But what if an exceptions is thrown?

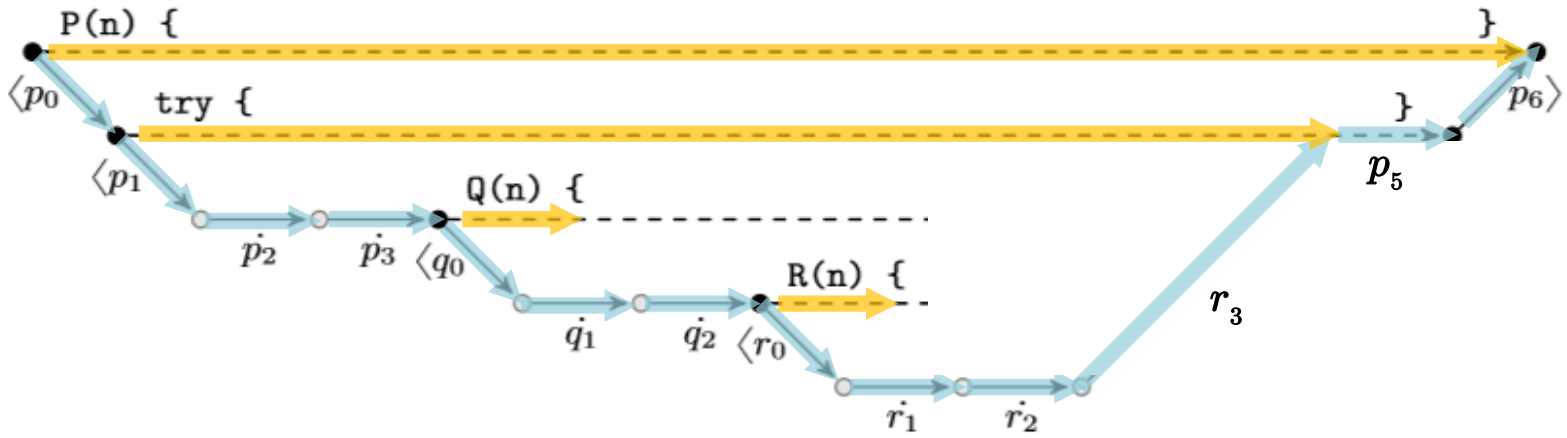
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```



Weakness of Nested Words

- But what if an exceptions is thrown?

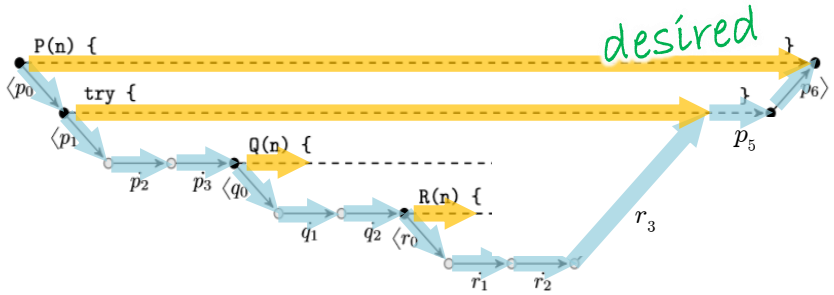
```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```

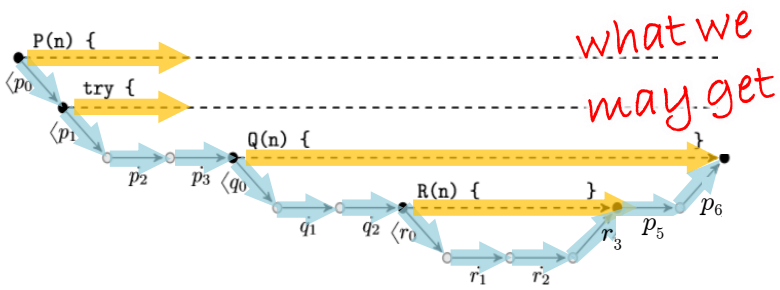


Weakness of Nested Words

- But what if an exceptions is thrown?

```
p0: P(n) {                q0: Q(n) {              r0: R(n) {
p1:     try {             q1:     y = n / 2       r1:     z = n-1
p2:         x = 2*n       q2:     y = R(y)        r2:     if (z<0)
p3:         x = Q(x)      q3:     y = y+1         r3:         throw 0
p4:     }                 q4:     return y }      r4:     return z }
p5:     catch (int) {}
p6:     return x }
```





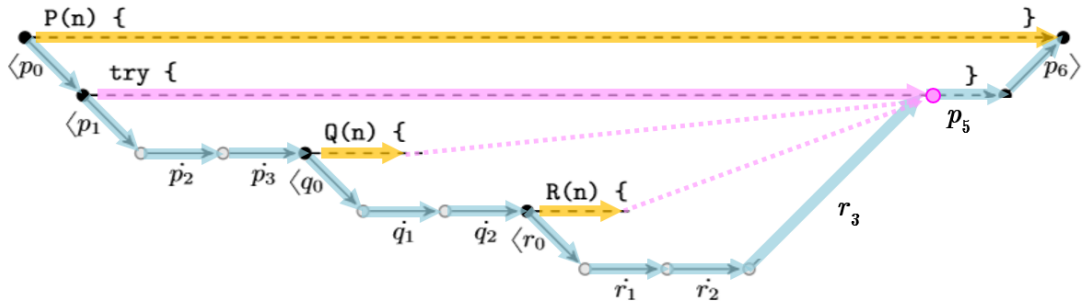
Other Examples – Python program

```
1   2    3    4    5
def isPrime(n):
    if n<2: return False
    elif n==2: return True
    else:
        i = 2
        while i < sqrt(n)+1:
            if n%i==0:
                return False
            else:
                i = i+1
    return True
```

Closes scopes 5,4,3

Colored Nested Words - Encoding

- **Explicitly**, using a graph with several types of edges (linear, mono-chromatic and bi-chromatic) and defining the constraints they should satisfy.



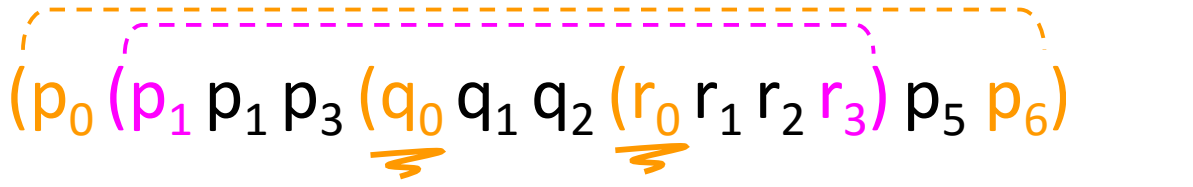
- **Implicitly**, using colored/types parenthesis.

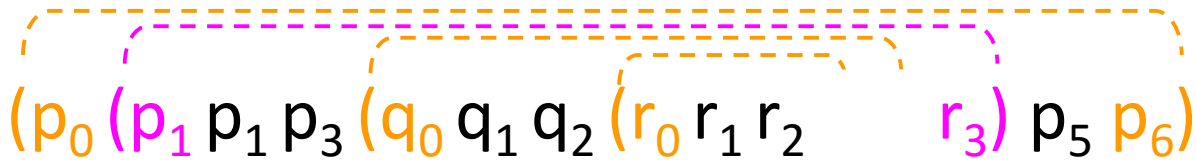
$(p_0\,(p_1\,p_1\,p_3\,(q_0\,q_1\,q_2\,(r_0\,r_1\,r_2\,r_3)\,p_5\,p_6)$

$[p_0\,(p_1\,p_1\,p_3\,[q_0\,q_1\,q_2\,[r_0\,r_1\,r_2\,r_3)\,p_5\,p_6]$

Regularity

- How can we define **regularity** of a language of colored nested words?
- Intuitively, we'd like the language obtained by inserting the missing closing tags to be a well-colored regular language of nested words.





Regularity

- Suppose we have such a mapping from CNW to NW

  f(w) = w'

  such that w' has all the missing closing tags

- Definition:

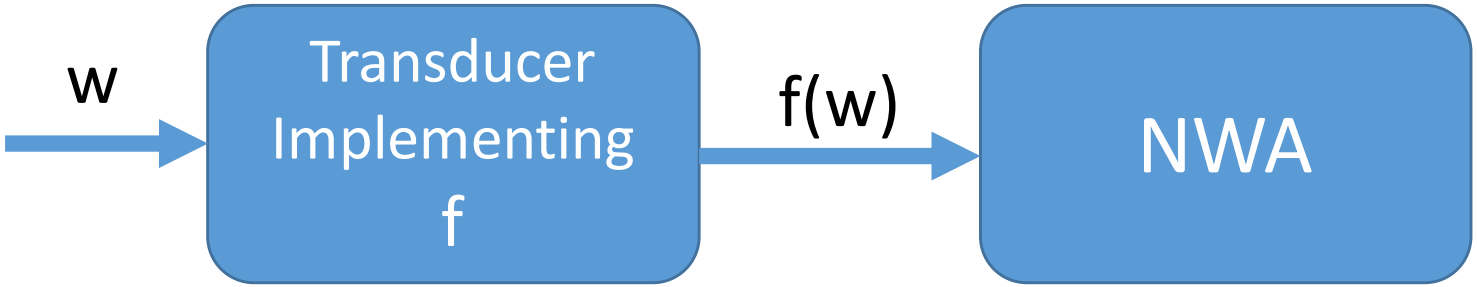
  Let L be a language of CNW.

  Then L is **regular** if

  { f(w) | w∈L } is a regular lang. of NW

Alur & Fisman

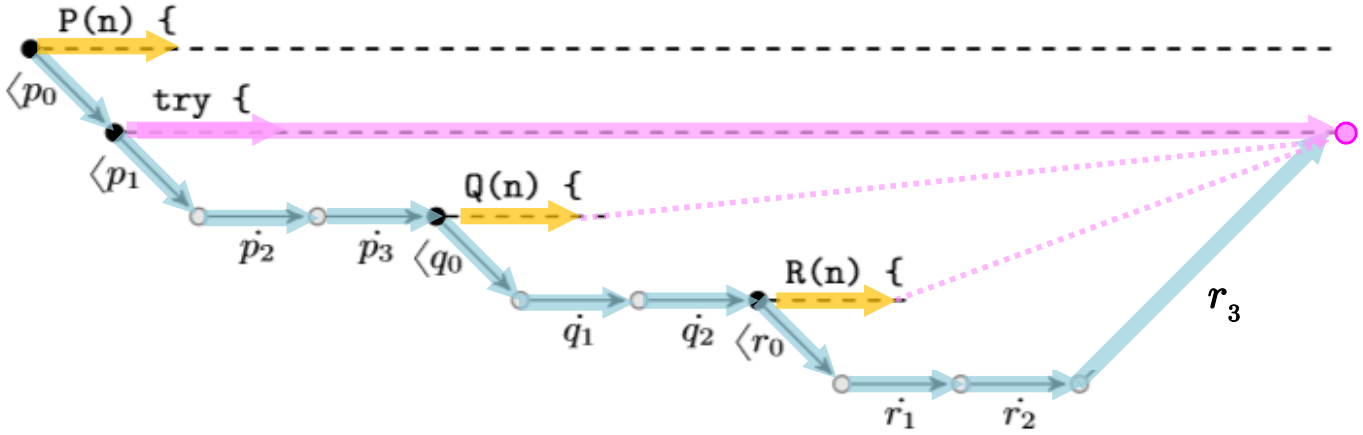
How can we process regular CNW ?



- But how can we implement this transducer?
- It is not regular, and not regular NW, because it needs to pop more than one letter when reading a “rescuing closing tag”!

Colored Nested Words Automata

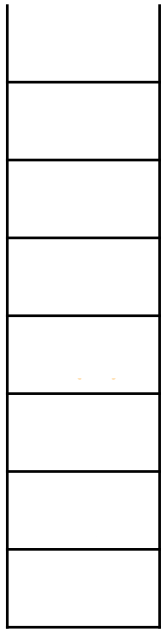
- The pushdown automaton marks the pushed letters with the respective color
- A read of an x-colored close-tag pops from the stack all symbols of y-colored symbols up to the first x-colored symbol (as well as this letter itself)

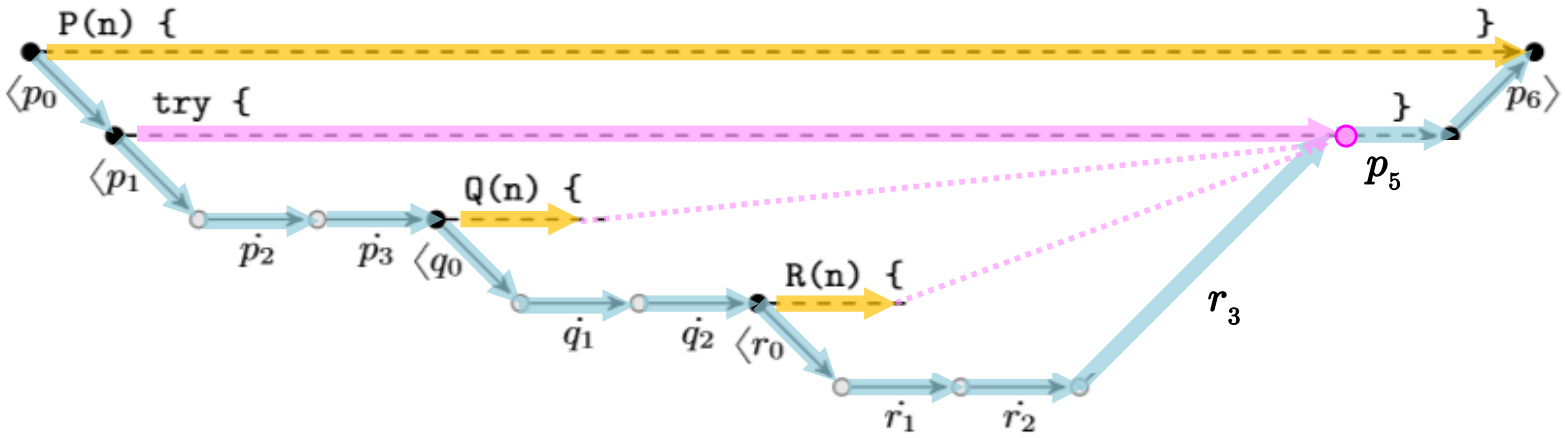


Colored Nested Words Automata

Alur & Fisman

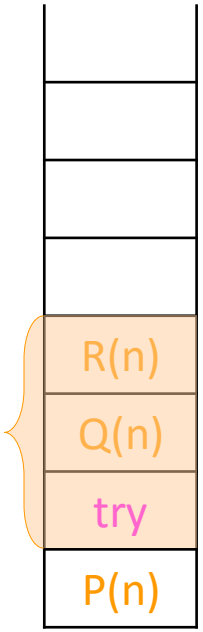
- The pushdown automaton marks the pushed letters with the respective color
- A read of an x-colored close-tag pops from the stack all symbols of y-colored symbols up to the first x-colored symbol (as well as this letter itself)





Blind vs. Sighted CNA

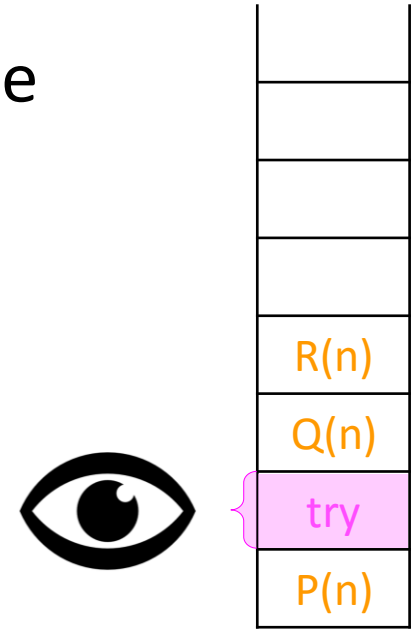
- Can the automaton view all the letters in the stack above the matching x-colored letter?
- Or can it just see that x-colored letter?
- We call the first version Sighted CNA



makes ε-transitions when popping y-colored letters

Blind vs. Sighted CNA

- Can the automaton view all the letters in the stack above the matching x-colored letter?
- Or can it just see that x-colored letter?
- We call the first version Sighted CNA
- And the second version Blind CNA
- Theorem 1



Sighted CNA and Blind CNA
have the same expressive power

Blind vs. Sighted CNA

- Theorem 2

Sighted CNA and Blind CNA
accept all the regular languages
of colored nested words

Other Examples – broken html

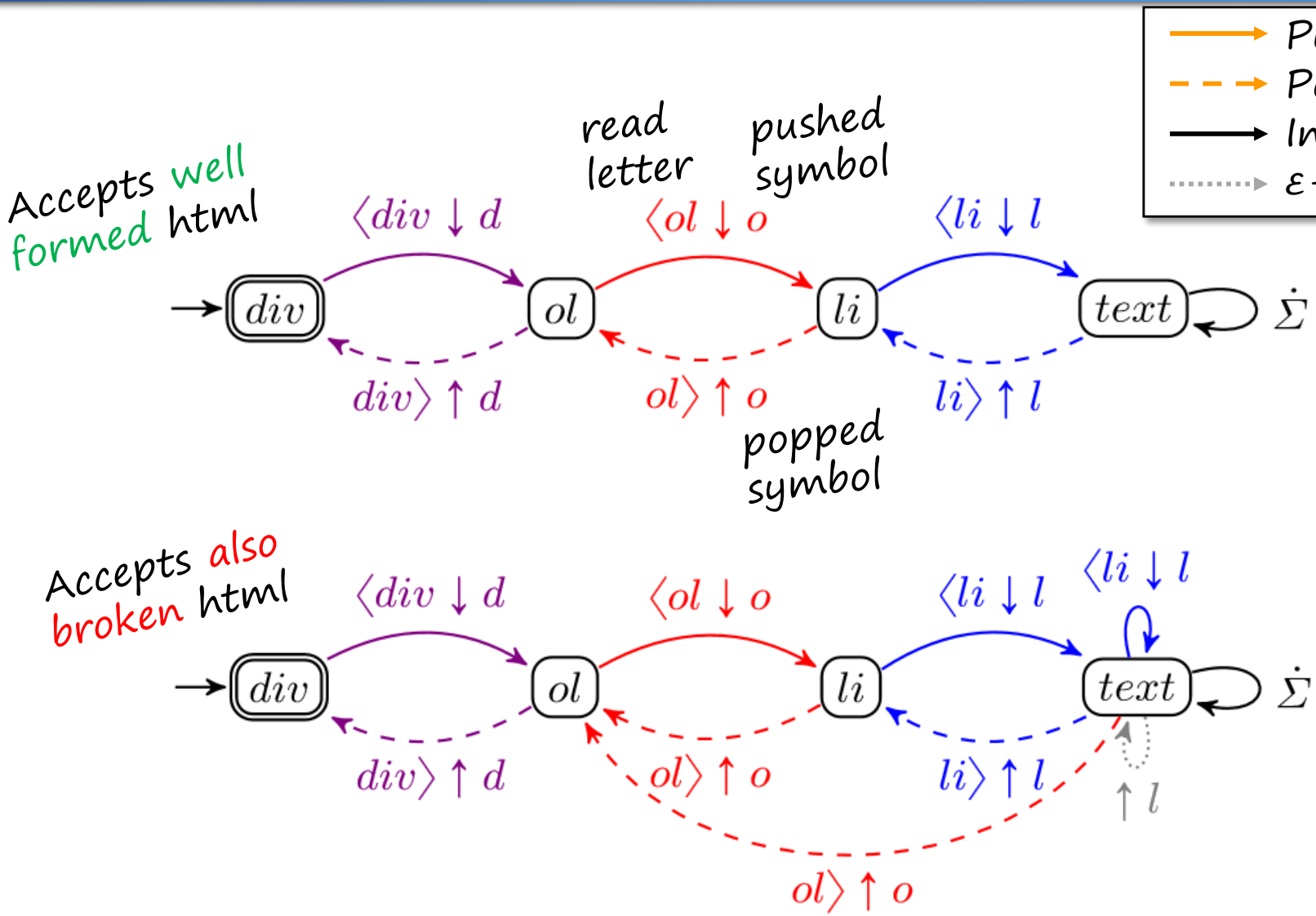
```
<div id="s">
    <ol>
            <li> ML  </li>
            <li> Haskell </li>
            <li> Scheme
    </ol>
</div>
<div id="w">
    <ol>
            <li> C
            <li> C++ </li>
            <li> Perl </li>
    </ol>
</div>
```

Missing closing tag

Missing closing tag

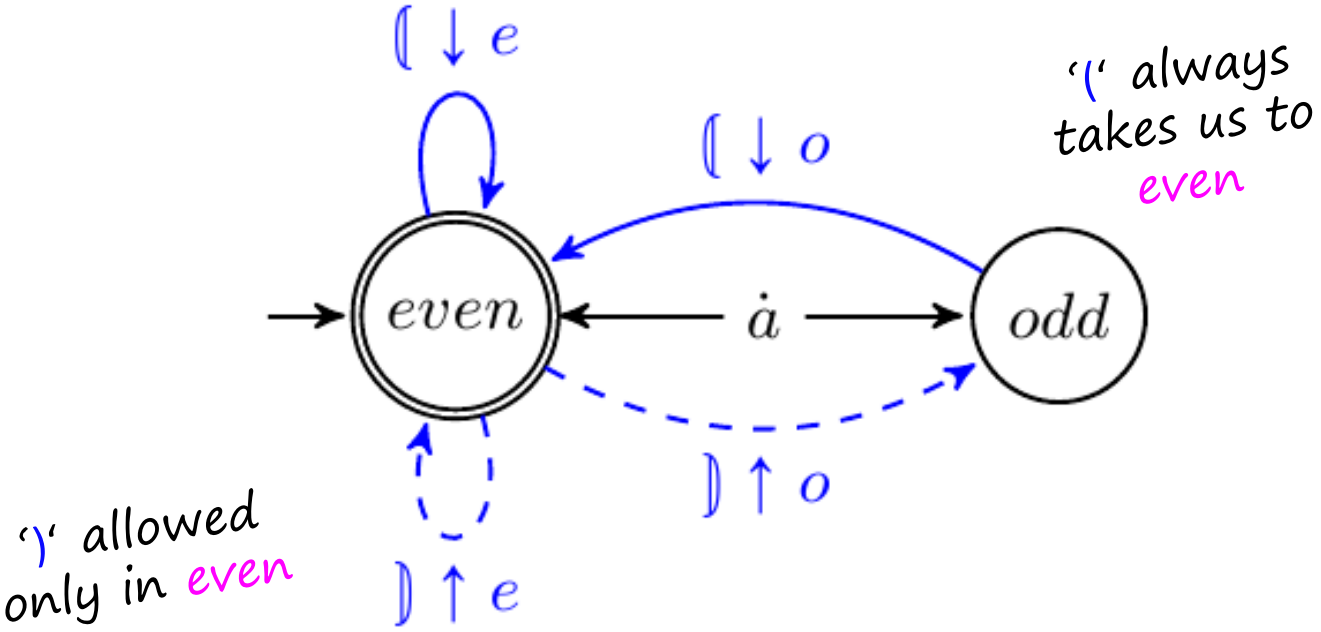
Alur & Fisman

CNA – html example



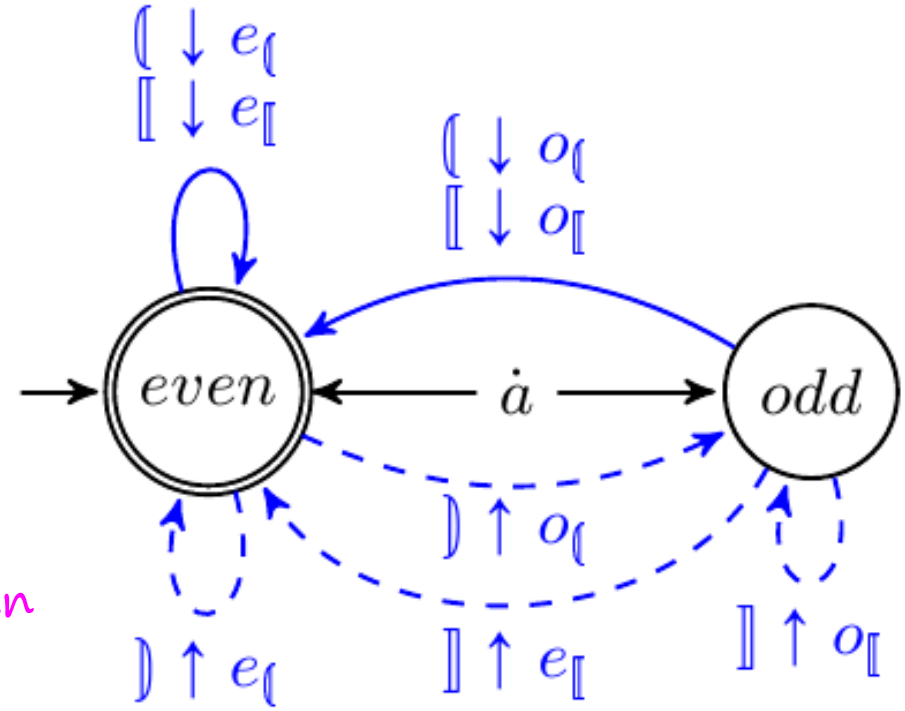
CNA - Examples

Accepts word where #a's between () is even



CNA - Examples

Accepts word where #a's between () is even
and #a's between [] is odd



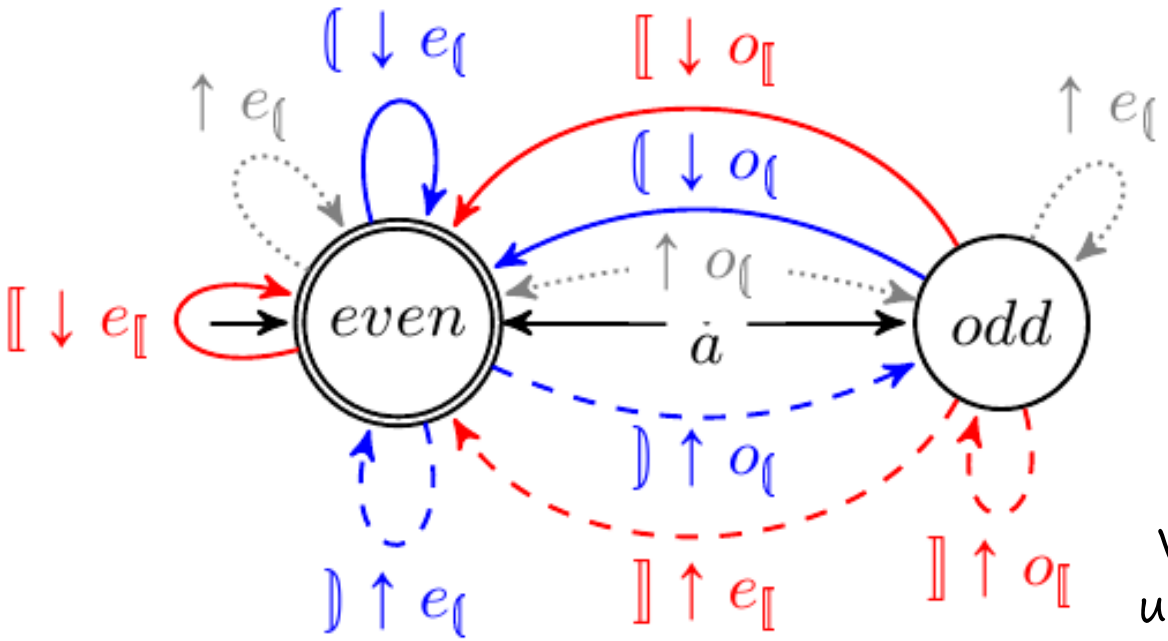
Also need to record the type of open tag '(' / '['

')' allowed only in even

']' allowed only in odd

CNA - Examples

Accepts word where #a's between () is even
and #a's between [] is odd
a (may be unmatched if encapsulated by []



We need different colors for '()' and '[]'

We can pop unmatched '('

Properties of Regular Languages of Colored Nested Words Automata

Alur & Fisman

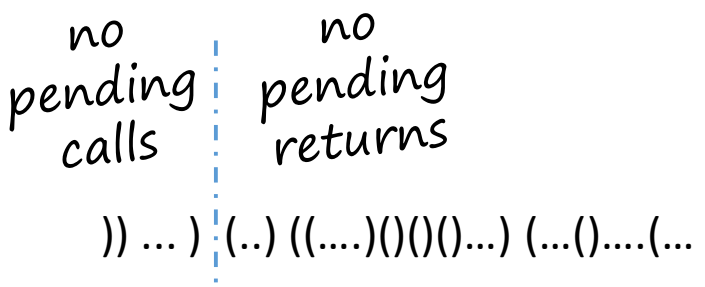
- Deterministic nested word automata are as expressive as their non-deterministic counterparts

- The class is closed under:
  - union
  - intersection
  - complementation
  - concatenation
  - Kleene-*
  - prefixes
  - ~~reversal~~ not closed under reversal
  - homomorphism

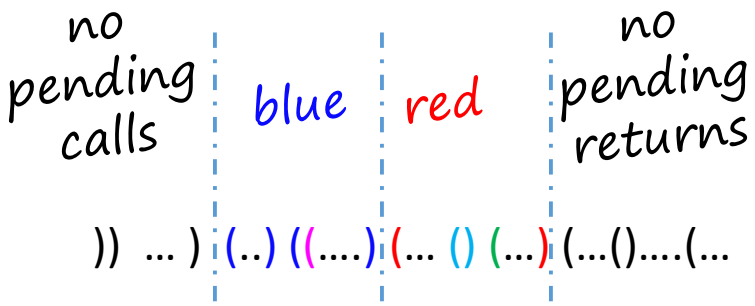
- The following problems are decidable:
  - emptiness
  - membership
  - inclusion
  - equivalence

Grammar Characterization

- There exists a grammar characterization for NWs
  - It classifies variables to two sets
  - One that disallows pending calls
  - One that disallows pending returns



- There exists a grammar characterization for CNWs
  - It classifies variables |C|+2 sets
  - One per each color
  - One that disallows pending calls
  - One that disallows pending returns



Thank you for your attention !

Comments / Questions ?

Rajeev Alur and Dana Fisman