Parsing with constraint

```
E ::= E+E
E ::= E*E
E ::= E=E
E ::= Num | true | false
```

Parse:

1=2*3=true

Lec10: A CYK for Any Grammar Would Do This

input: grammar G, non-terminals $A_1,\ldots,A_K$, tokens $t_1,\ldots.t_L$

word: $\mathbf{w} \equiv \mathbf{w_{(0)} w_{(1)} \ldots w_{(N-1)}}$

notation: $w_{p..q} = w_{(p)} w_{(p+1)} \ldots w_{(q-1)}$

output: P set of **(A, i, j)** implying $\mathbf{A \Rightarrow^* w_{i..j}}$, A can be: A_k, t_k, or ε

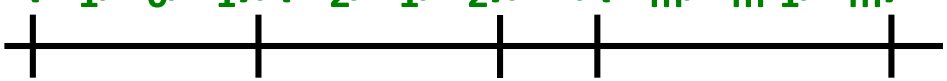
$P = \{(w_{(i)},i,i+1) \mid 0 \leq i < N-1\}$

repeat {

choose rule $(A::=B_1\ldots B_m) \in G$

if $((A,k_0,k_m) \notin P$ && (for some $k_1,\ldots,k_{m-1}$:

$((m=0$ && $k_0=k_m)$ || $(B_1,k_0,k_1),(B_2,k_1,k_2),\ldots,(B_m,k_{m-1},k_m) \in P)))$

$P := P \cup \{(A,k_0,k_m)\}$ 

} until no more insertions possible

What is the maximal number of steps?
How long does it take to check step for a rule?
} for a given grammar

```
1=2*3=true
0123456   7
```

1. E ::= E+E
2. E ::= E*E
3. E ::= E=E
4. E ::= Num | true | false

(Num, 0, 1)
(=, 1, 2)
(Num, 2, 3)
(*, 3, 4)
(Num, 4, 5)
(=, 5, 6)
(true, 6, 7)

4: (E, 0, 1)
4: (E, 2, 3)
4: (E, 4, 5)
4: (E, 6, 7)

3: (E, 0, 3)
2: (E, 2, 5)
3: (E, 4, 7)

2, 3: (E, 0, 5)
3, 2: (E, 2, 7)

3, 2, 3: (E, 0, 7)

Parse trees

(1=(2*3))=true

((1=2)*3)=true

(1=2)*(3=true)

1=(2*(3=true))

1=((2*3)=true)

Which one are correct according to:

+ : Int x Int→Int

* : Int x Int→Int

= : Bool x Bool → Bool

= : Int x Int → Bool

Parse trees

(1=(2*3))=true : Correct

((1=2)*3)=true : Bool*Int incorrect

(1=2)*(3=true) : Bool*Bool incorrect

1=(2*(3=true)) : Int*Bool incorrect

1=((2*3)=true) : Int=Bool incorrect

Lec10: A CYK Algorithm Producing Results

Rule **$(A::=B_1...B_m\ , f) \in G$** with **semantic action f**

$f : (R \cup T)^m \to R$ **R – results (e.g.trees)** **T - tokens**

Useful parser: returning a set of result (e.g. syntax trees)

((A, p, q),**r**): $A \Rightarrow^* w_{p..q}$ **and** the result of parsing is **r**

$P = \{((A,i,i+1), f(w_{(i)})) \mid 0 \le i < N-1\ \&\&\ ((A ::= w_{(i)}),f) \in G)\}$ // unary

repeat {

choose rule $(A::=B_1B_2\ , f) \in G$

if $((A,k_0,k_2) \notin P\ \&\&$ for some k_1: $((B_1,k_0,k_1),r_1), ((B_2,p_1,p_2),r_2) \in P$

$P := P \cup \{((A,k_0,k_2), f(r_1,r_2))\}$

} until no more insertions possible

Compute parse trees using identity functions as semantic actions:

$((A ::= w_{(i)})$, $x:R \Rightarrow x)$ **$((A::=B_1B_2)$, $(r_1,r_2):R^2 \Rightarrow Node_A(r_1,r_2)$)**

A bound on the number of elements in **P**? 2^N : squared in each level

Lec 10: A CYK Algorithm with Constraints

Rule $(A::=B_1...B_m , f) \in G$ with **partial function** semantic action **f**

f : (R∪T)^m -> Option[R] **R – results** **T - tokens**

Useful parser: returning a set of results (e.g. syntax trees)

$((A, p, q), \mathbf{r})$: $A =>^* w_{p..q}$ **and** the result of parsing is $\mathbf{r} \in \mathbf{R}$

$P = \{((A,i,i+1), f(w_{(i)}).get) \mid 0 \leq i < N-1$ && $((A ::= w_{(i)}), f) \in G)\}$

repeat {

choose rule $(A::=B_1B_2 , f) \in G$

if $((A,k_0,k_2) \notin P$ && for some k_1: $((B_1,k_0,k_1),r_1), ((B_2,p_1,p_2),r_2) \in P$

and $f(r_1,r_2)$!= None //apply rule only if **f** is defined

$P := P \cup \{((A,k_0,k_2), f(r_1,r_2).get)\}$

} until no more insertions possible

```
sealed trait Tree
sealed trait IntTree extends Tree
sealed trait BoolTree extends Tree
case class Times(e1: IntTree, e2: IntTree) extends IntTree
case class Plus(e1: IntTree, e2: IntTree) extends IntTree
case class Equals(e1: Tree, e2: Tree) extends BoolTree
case class Num(i: Int) extends IntTree
case object True extends BoolTree // Same for false
```

1. E ::= E+E (e1, _, e2) => mkPlus(e1, e2)
2. E ::= E*E (e1, _, e2) => mkTimes(e1, e2)
3. E ::= E=E (e1, _, e2) => mkEquals(e1, e2)
4. E ::= Num | true | false e => mkLit(_)

```scala
sealed trait Tree
sealed trait IntTree extends Tree
sealed trait BoolTree extends Tree
case class Times(e1: IntTree, e2: IntTree) extends IntTree
case class Plus(e1: IntTree, e2: IntTree) extends IntTree
case class Equals(e1: Tree, e2: Tree) extends BoolTree
case class Num(i: Int) extends IntTree
case object True extends BoolTree // Same for false
```

```scala
def mkTimes(e1 : Tree, e2: Tree) : Option[Tree] = (e1, e2) match {
  case (t1: IntTree, t2: IntTree) => Some(Times(t1, t2))
  case _ => None
} // Same for Plus
```

```scala
def mkEquals(e1 : Tree, e2: Tree) : Option[Tree] = (e1, e2) match {
  case (t1: IntTree, t2: IntTree) => Some(Equals(t1, t2))
  case (t1: BoolTree, t2: BoolTree) => Some(Equals(t1, t2))
  case _ => None
}
```

`1=2*3=true`

`0123456   7`

1. E ::= E+E
2. E ::= E*E
3. E ::= E=E
4. E ::= Num | true | false

(Num, 0, 1)
(=, 1, 2)
(Num, 2, 3)
(*, 3, 4)
(Num, 4, 5)
(=, 5, 6)
(true, 6, 7)

4: (E, 0, 1, Num)
4: (E, 2, 3, Num)
4: (E, 4, 5, Num)
4: (E, 6, 7, Num)

3: (E, 0, 3, Equals(N, N))
2: (E, 2, 5, Times(N, N))

3: (E, 0, 5, Equals(N, Times(N ,N)))

3: (E, 0, 7, Equals(Equals(N, Times(N, N)), True))

Type checking

bit.ly/16CF6v2

```
def swap(lst: Array[Int], a: Int, b: Int): Array[Int] = {
  if (a >= lst.length || b >= lst.length) lst else {
    val swap = lst(a)
    lst(a) = lst(b)
    lst(b) = swap
    lst
  }
} // lst(a) = … ⇔ lst.update(a, …)
// lst(a) ⇔ lst.apply(a)
```

$\Gamma^{Array[Int]}$={(length, Int), (apply: Array[Int] x Int → Int), (update: Array[Int] x Int x Int → void)}

Γ^{Int}={(>=, Int x Int → Bool)}

Γ^{Bool}={(||, Bool x Bool → Bool)}

Γ^{top}={(swap, T x Array[Int] x Int x Int → Int)}

Γ^{swap}={(lst, Array[Int]), (a, Int), (b, Int)} U Γ^{top}

$\Gamma^{Array[Int]}$={(length, Int), (apply: Array[Int] x Int → Int), (update: Array[Int] x Int x Int → void)}

Γ^{Int}={(>=, Int x Int → Bool)}

Γ^{top}={(swap, T x Array[Int] x Int x Int → Int)}

Γ^{swap}={(lst, Array[Int]), (a, Int), (b, Int)} U Γ^{top}

$\Gamma^{swap} \vdash$ a >= lst.length || b >= lst.length : Bool

$\Gamma^{swap} \vdash$ lst : Array[Int]

$\Gamma^{swap} \vdash$ {val swap = lst(a); lst(a) = lst(b); lst(b) = swap; lst } : Array[Int]

$\Gamma^{swap} \vdash$ if (a >= lst.length || b >= lst.length) lst else { val swap = lst(a); lst(a) = lst(b); lst(b) = swap; lst } : Array[Int]

$$\frac{\Gamma^{swap} \vdash a : Int \qquad \Gamma^{Int} \vdash \texttt{>=} : Int \times Int \to Bool \qquad \dfrac{\Gamma^{swap} \vdash lst : Array[Int] \qquad \Gamma^{Array} \vdash length : Int}{\Gamma^{swap} \vdash lst.length : Int}}{\Gamma^{swap} \vdash a \texttt{ >= } lst.length : Bool}$$

$$\frac{\Gamma^{swap} \vdash a \texttt{ >= } lst.length : Bool \qquad \Gamma^{Bool} \vdash \texttt{||} : Bool \times Bool \to Bool \qquad \Gamma^{swap} \vdash b \texttt{ >= } lst.length : Bool}{\Gamma^{swap} \vdash a \texttt{ >= } lst.length \texttt{ || } b \texttt{ >= } lst.length : Bool}$$

$\Gamma^{Array[Int]}$={(length, Int), (apply: Array[Int] x Int **->** Int), (update: Array[Int] x Int x Int -> void)}

Γ^{Int}={(>=, Int x Int -> Bool)}

Γ^{Bool}={(||, Bool x Bool → Bool)}

Γ^{top}={(swap, T x Array[Int] x Int x Int -> Int)}

Γ^{swap}={(lst, Array[Int]), (a, Int), (b, Int)} U Γ^{top}

$\Gamma^{\text{Array[Int]}}$={(length, Int), (apply: Array[Int] x Int **->** Int), (update: Array[Int] x Int x Int -> void)}

Γ^{Int}={(>=, Int x Int -> Bool)}

Γ^{top}={(swap, T x Array[Int] x Int x Int -> Int)}

Γ^{swap}={(lst, Array[Int]), (a, Int), (b, Int)} U Γ^{top}

Γ^{swap2}={(lst, Array[Int]), (a, Int), (b, Int), (swap, Int)}

$$\frac{\Gamma^{\text{swap}} \vdash \text{lst : Array[Int]} \qquad \Gamma^{\text{Array[Int]}} \vdash \text{apply : Array[Int] x Int -> Int} \qquad \Gamma^{\text{swap}} \vdash \text{a : Int}}{\Gamma^{\text{swap}} \vdash \text{lst(a) : Int}}$$

$$\Gamma^{\text{swap}} \vdash \text{lst(b) : Int}$$

$$\frac{\dfrac{\dfrac{\Gamma^{\text{swap2}} \vdash \text{lst(b): Int} \quad \Gamma^{\text{swap2}} \vdash \text{lst: Array} \quad \Gamma^{\text{swap2}} \vdash \text{a : Int} \quad \Gamma^{\text{Array[Int]}} \vdash \text{update: Array[Int] x Int x Int -> void}}{\Gamma^{\text{swap2}} \vdash \text{lst(a) = lst(b) : void}} \qquad \Gamma^{\text{swap2}} \vdash \text{\{lst(b) = swap; lst \} : Array[Int]}}{\Gamma^{\text{swap2}} \vdash \text{\{lst(a) = lst(b); lst(b) = swap; lst \} : Array[Int]}} \qquad \Gamma^{\text{swap}} \vdash \text{lst(a) : Int}}{\Gamma^{\text{swap}} \vdash \text{\{val swap = lst(a); lst(a) = lst(b); lst(b) = swap; lst \} : Array[Int]}}$$

$\Gamma^{Array[Int]}$={(length, Int), (apply: Array[Int] x Int **->** Int), (update: Array[Int] x Int x Int -> void)}
Γ^{Int}={(>=, Int x Int -> Bool)}
Γ^{top}={(swap, T x Array[Int] x Int x Int -> Int)}
Γ^{swap}={(lst, Array[Int]), (a, Int), (b, Int)} U Γ^{top}
Γ^{swap2}={(lst, Array[Int]), (a, Int), (b, Int), (swap, Int)}

$$\dfrac{\dfrac{\Gamma^{swap2} \vdash \text{swap: Int} \quad \Gamma^{swap2} \vdash \text{lst: Array} \quad \Gamma^{swap2} \vdash \text{b : Int} \quad \Gamma^{Array[Int]} \vdash \text{update: Array[Int] x Int x Int -> void}}{\Gamma^{swap2} \vdash \text{lst(b) = swap : void} \qquad \Gamma^{swap2} \vdash \text{lst : Array[Int]}}}{\Gamma^{swap2} \vdash \text{\{lst(b) = swap; lst \} : Array[Int]}}$$