

Introduktion till grammatik

TDP019 Projekt: Datorspråk
Föreläsning 2

Översikt

- BNF-formalismen
- Grammatik
- Parsesträd
- Syntaxsträd
- Semikolon, begin-end eller inte
- Argumentlösa fuktionsanrop (med parentes?)

BNF

Backus-Naur form

$$A \rightarrow \alpha$$

BNF

Backus-Naur form

$$A \rightarrow \alpha$$

•

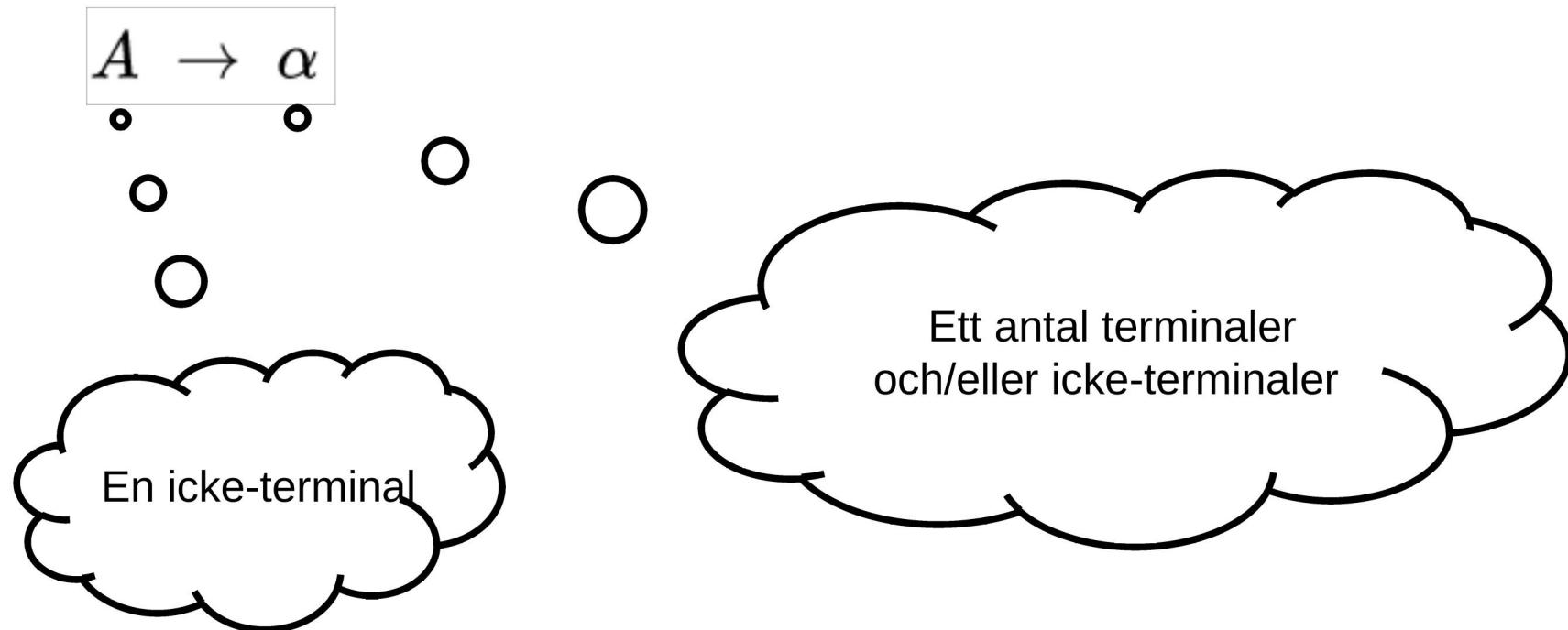
•

○



BNF

Backus-Naur form



BNF

Backus-Naur form

$$A \rightarrow \alpha$$

För att råda bot på stölkiga grammatiker

BNF – Helhet och delar

Helhet → Del Del Del

Helhet ← Del Del Del

BNF – asci-notation

Helhet → Del Del Del
Helhet ← Del Del Del



Helhet ::= Del Del Del
Helhet =:: Del Del Del

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> "+" <uttryck>

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> "+" <uttryck>

Icke-terminaler kan härledas vidare av en regel (som kanske kommer ovanför eller under den aktuella raden)

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> "+" <uttryck>

Icke-terminaler kan härledas vidare av en regel (som kanske kommer ovanför eller under den aktuella raden)

Terminaler är tecken eller delar som inte kan härledas vidare

Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>

Grammatik – Terminal / icke-terminal

uttryck -> tal

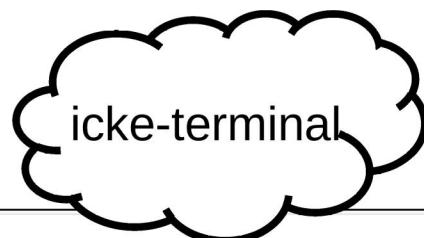
uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>
•

○

○



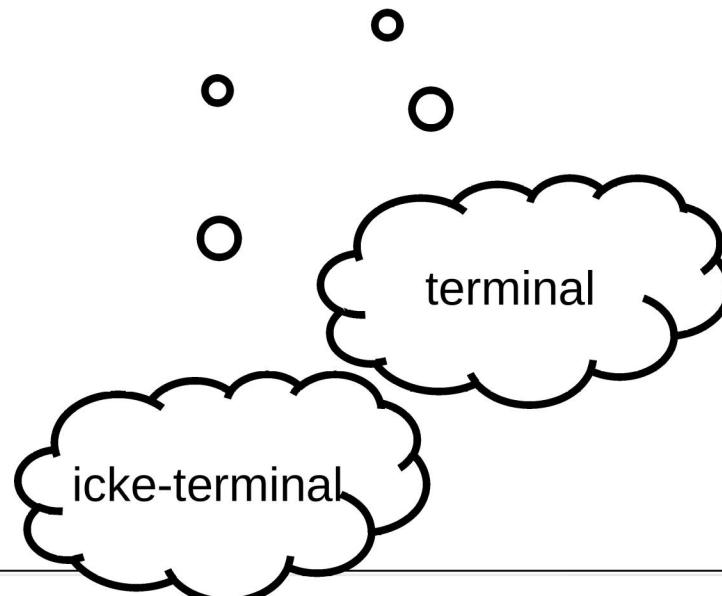
Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>



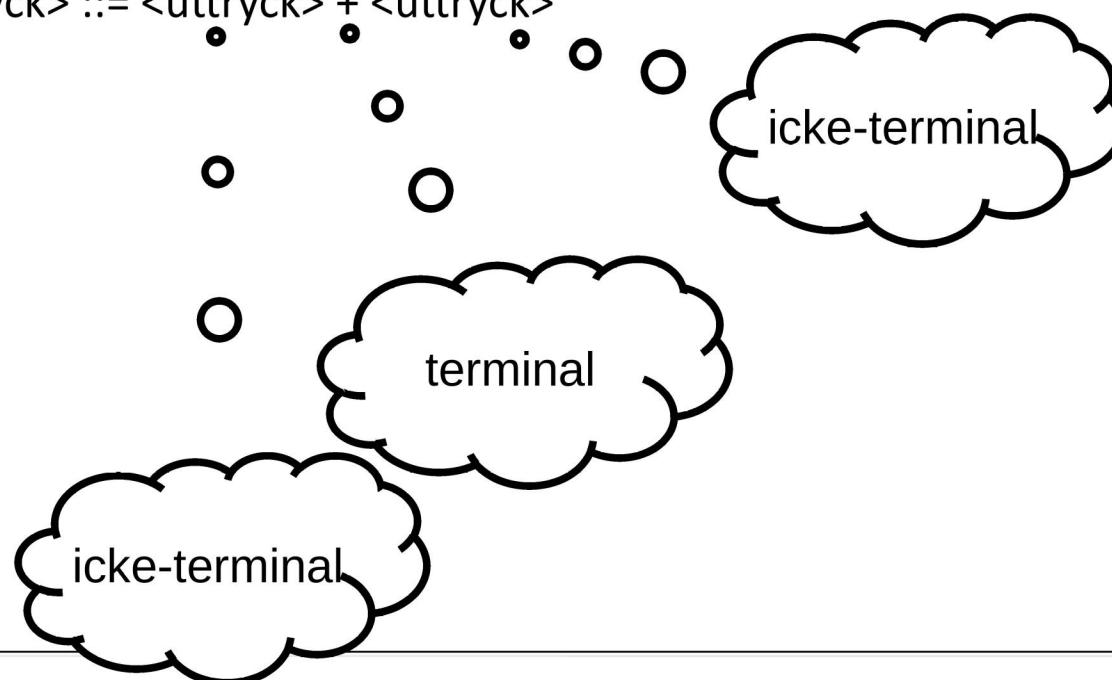
Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>



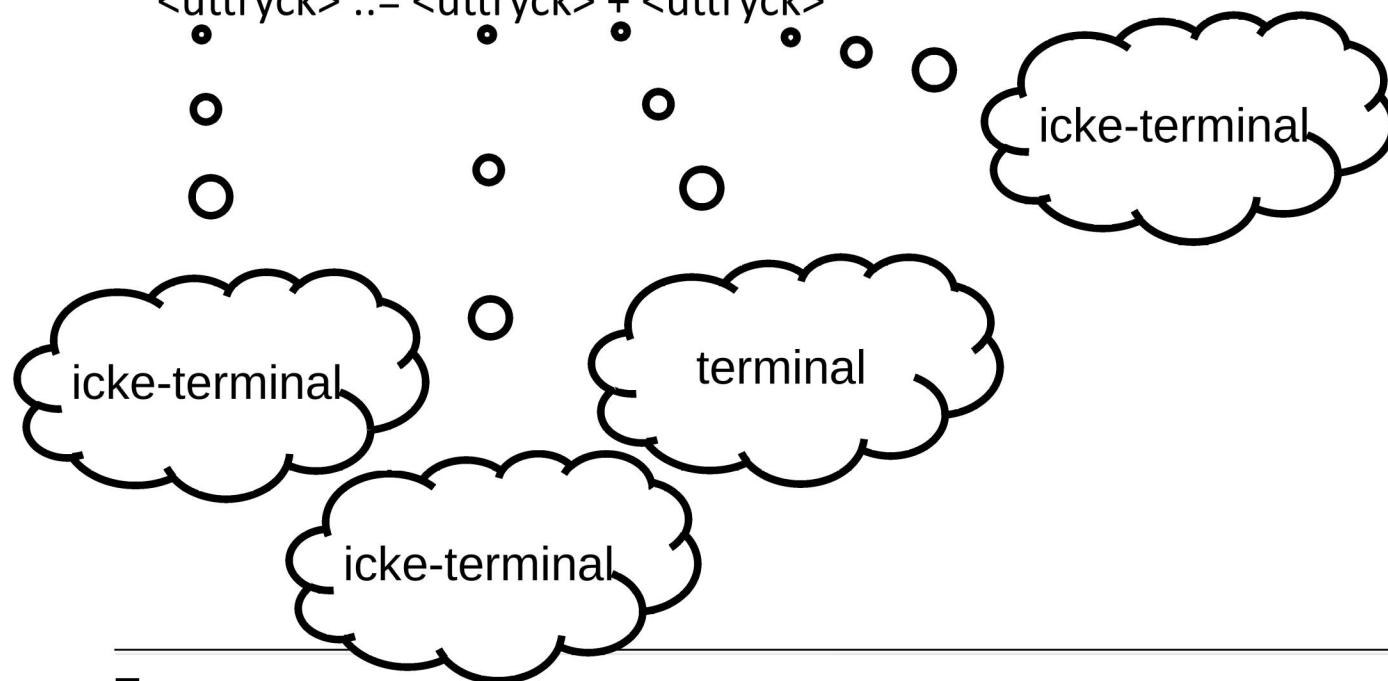
Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>



Grammatik – Terminal / icke-terminal

uttryck -> tal

uttryck -> uttryck plus uttryck

<uttryck> ::= <tal>

<uttryck> ::= <uttryck> + <uttryck>

Grammatik – adress

```
<postal-address> ::= <name-part> <street-address> <zip-part>  
  
<name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL> | <personal-part> <name-part>  
  
<personal-part> ::= <initial> "." | <first-name>  
  
<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>  
  
<zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>  
  
<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""  
<opt-apt-num> ::= <apt-num> | ""
```

John Lee Smith Sr.
132 My Street
Kingston, New York 12401

Grammatik

<program> ::= program <satser> slut-program

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
så <sats> | så <sats>
annars <sats> |

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
 så <sats> | så <sats>
 annars <sats> |

Om vi vill ha semikolon efter varje sats, hur fungerar det i vilkorsdelen?

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
 så <sats> | så <sats>
 annars <sats> |

Om vi vill ha semikolon efter varje sats, hur fungerar det i vilkorsdelen?
Behöver vi ha ett semikolon efter varje?

Grammatik

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
 så <sats> | så <sats>
 annars <sats> |

Om vi vill ha semikolon efter varje sats, hur fungerar det i vilkorsdelen?
Behöver vi ha ett semikolon efter varje?
Hur blir det om man utelämnar annars-delen?

Grammatik - end

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
så <sats> | så <sats>
annars <sats> | end
end

Grammatik - indentering

<program> ::= program <satser> slut-program

<satser> ::= <sats>
| <sats> <satser>

<sats> ::= <tilldelning> | <vilkor> | ... kan finnas massa mer, styrstrukturer etc

<tilldelning> ::= tilldela <variabel> <uttryck>

<vilkor> ::= om <predikatuttryck> | om <predikatuttryck>
så <sats> | så <sats>
annars <sats> |

Finns text om hur man hanterar indenteringskänslighet på kurswebben. Indentering är i princip bara en annan version av begin/end. Kräver lite eget läsande ansvar att få till det.

Parseträd

Parseträd

$$\begin{array}{l} E \rightarrow E + E \mid E * E \mid V \\ V \rightarrow x \mid y \mid z \end{array}$$

Parseträd – tolka ett uttryck

$$\begin{array}{l} E \rightarrow E + E \mid E * E \mid V \\ V \rightarrow x \mid y \mid z \end{array}$$

x * y + z

Parseträd – Lättare notation?

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z

Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z

Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z

Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

```
x * y + z
```

Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z



Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z



Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

x * y + z

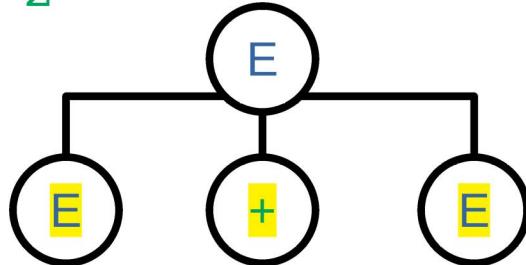


Varför $\langle E \rangle + \langle E \rangle$?

Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

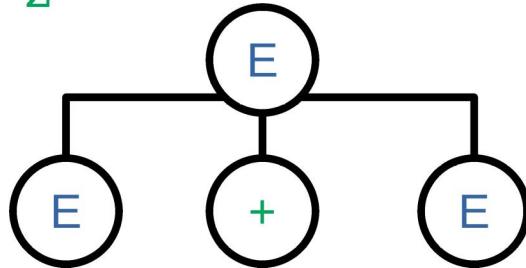
x * y + z



Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

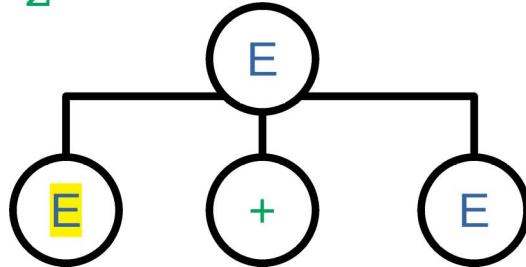
x * y + z



Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

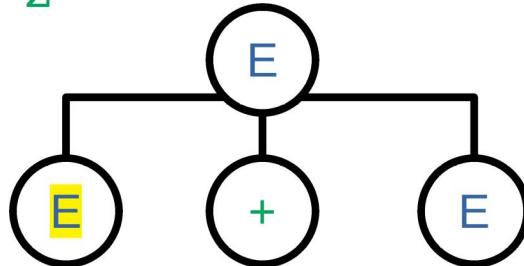
x * y + z



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

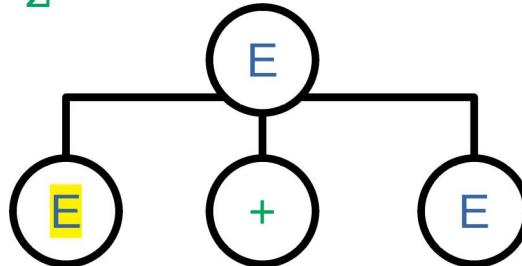
x * y + z



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

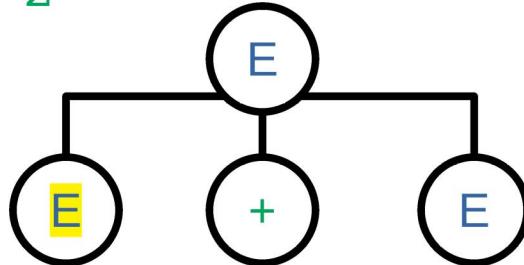
x * y + z



Parseträd – Bygg träd

<E> ::= <E> + <E> | <E> * <E> | <V>
<V> ::= x | y | z

x * y + z

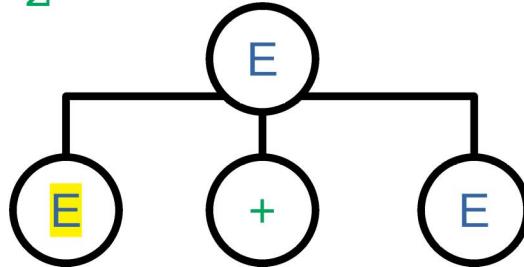


Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

$\langle V \rangle ::= x \mid y \mid z$

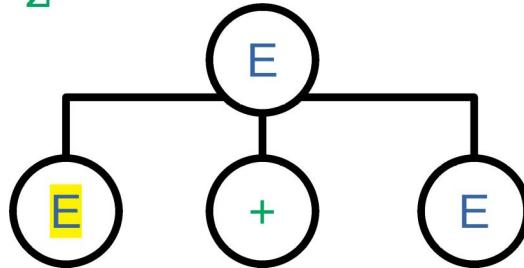
x * y + z



Parseträd – Bygg träd

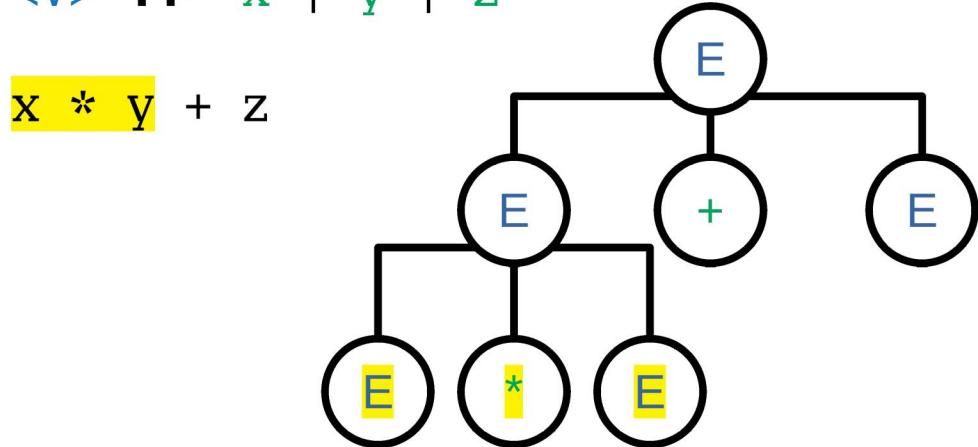
```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

x * y + z



Parseträd – Bygg träd

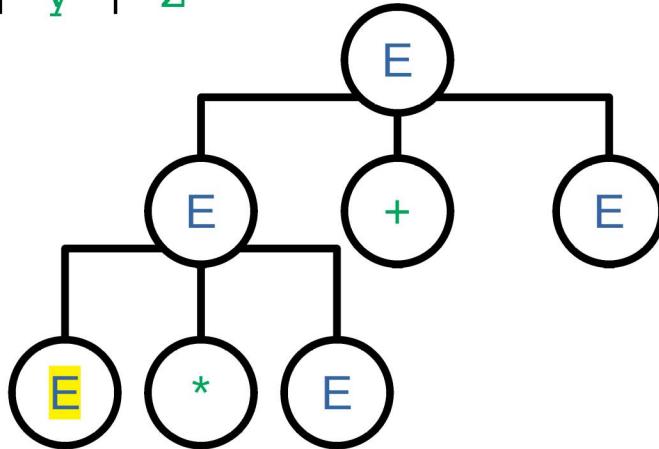
```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

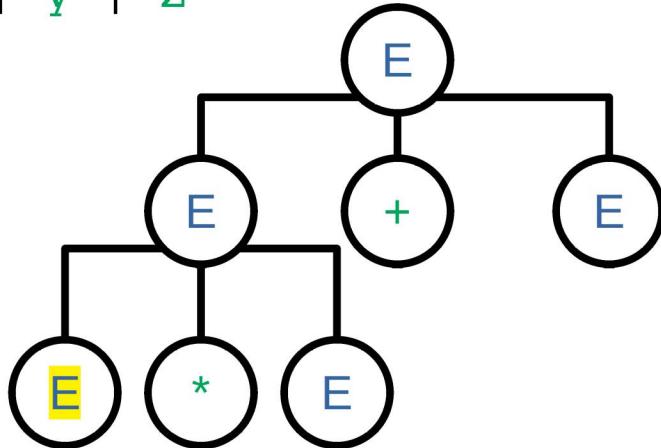
x * y + z



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

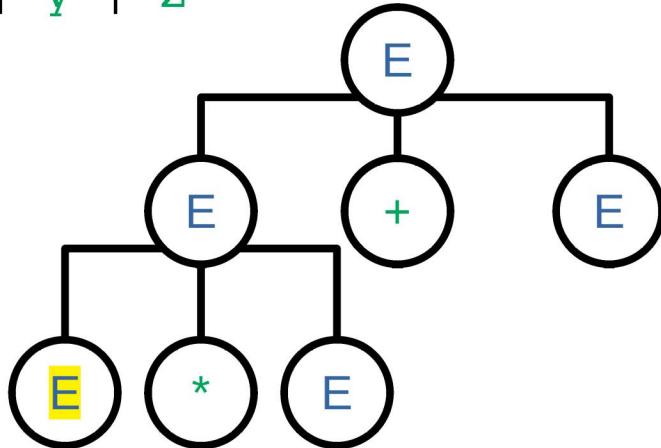
x * y + z



Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

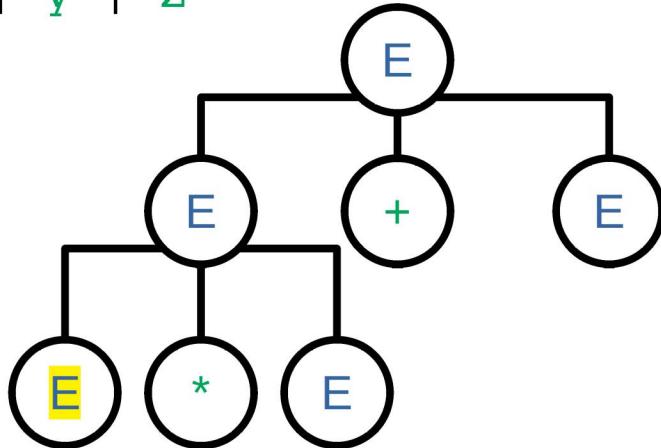
x * y + z



Parseträd – Bygg träd

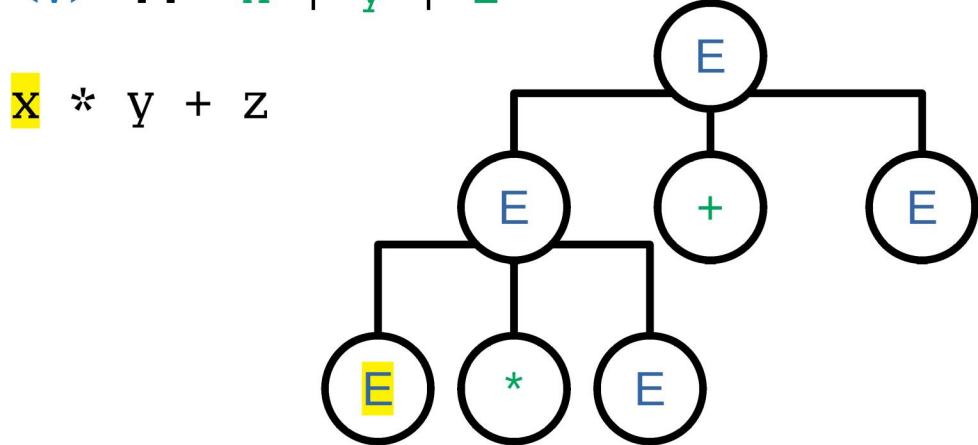
$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$

x * y + z



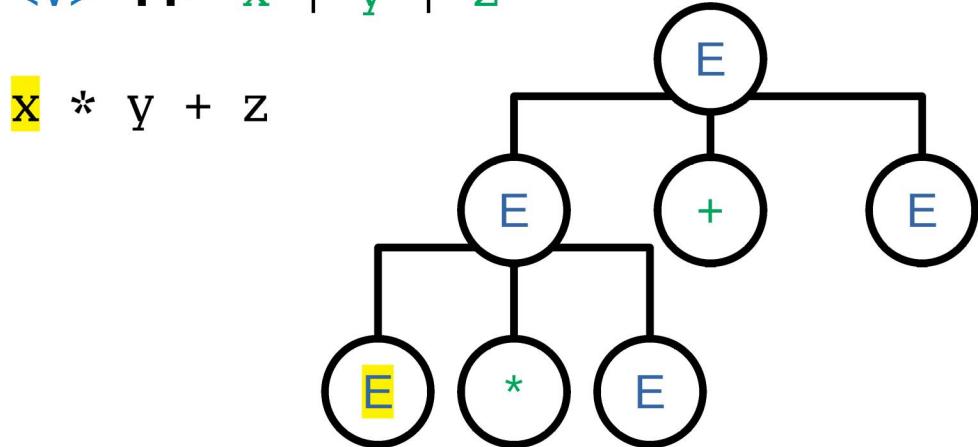
Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$



Parseträd – Bygg träd

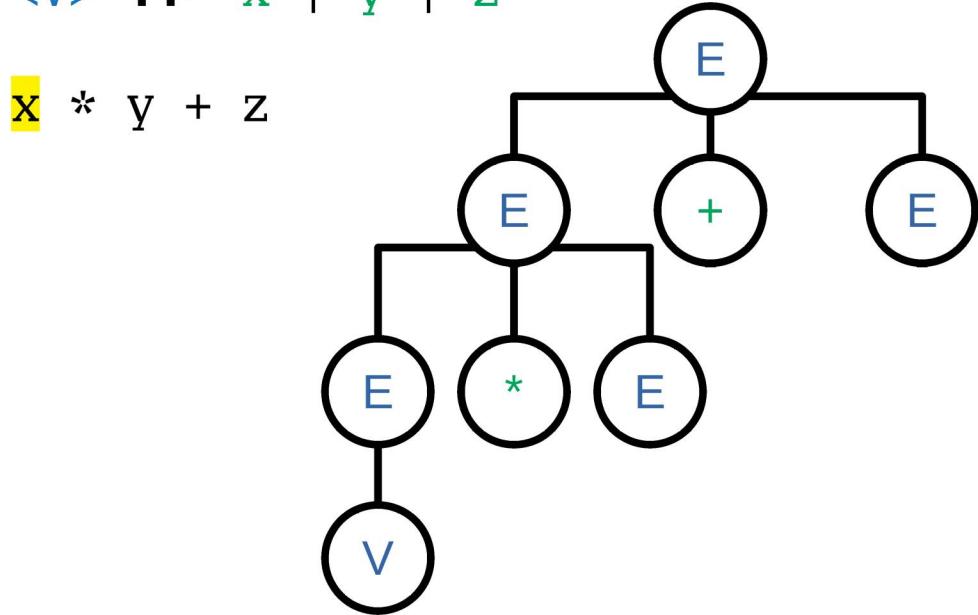
$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$



Parseträd – Bygg träd

57

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



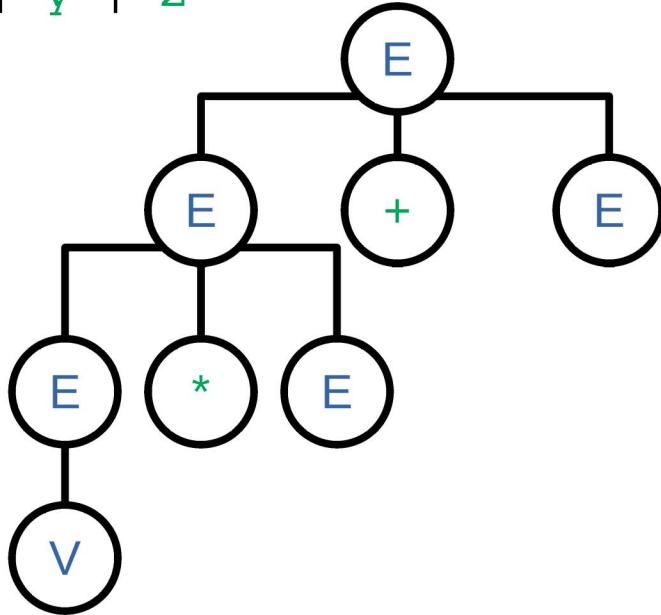
Parseträd – Bygg träd

58

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

$\langle V \rangle ::= x \mid y \mid z$

x * y + z

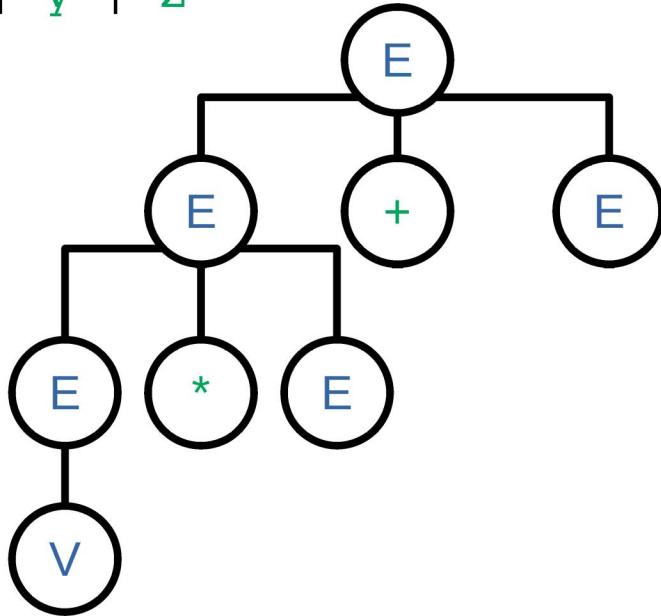


Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

$\langle V \rangle ::= x \mid y \mid z$

x * y + z

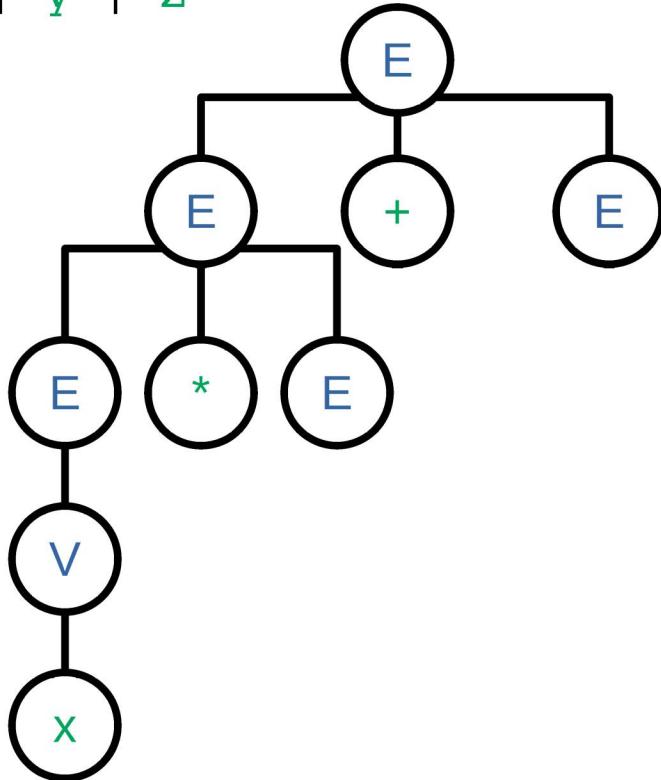


Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

$\langle V \rangle ::= x \mid y \mid z$

x * y + z

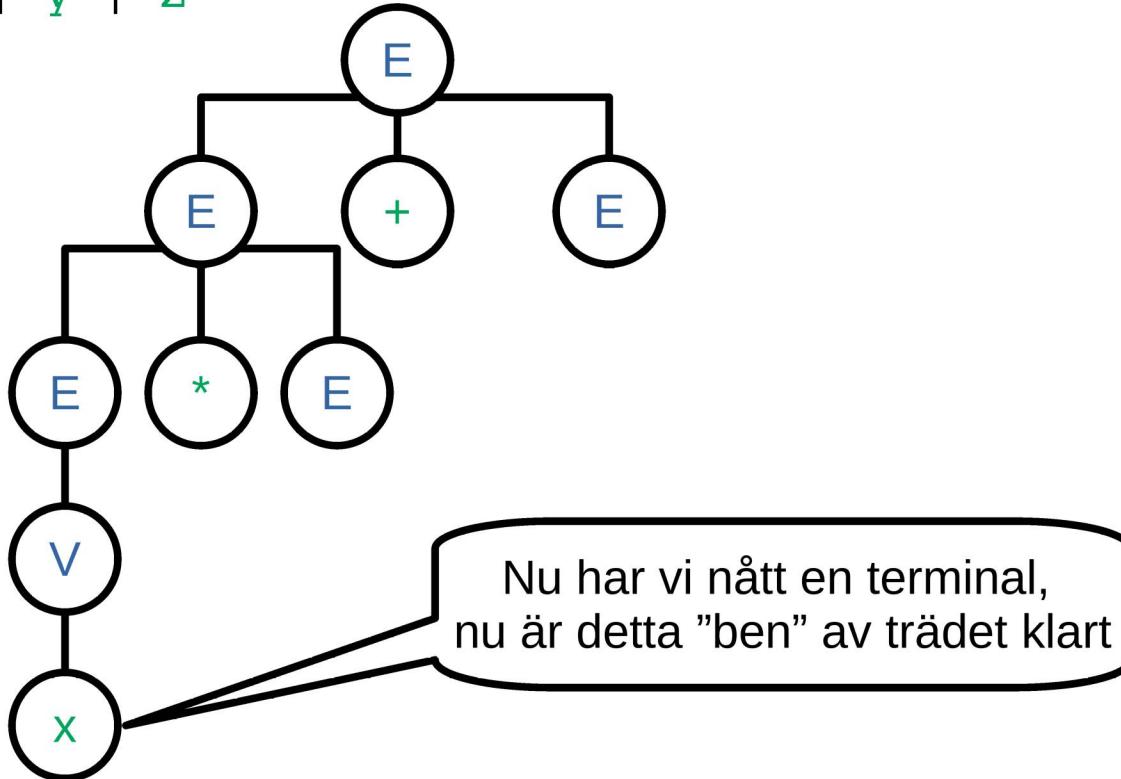


Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

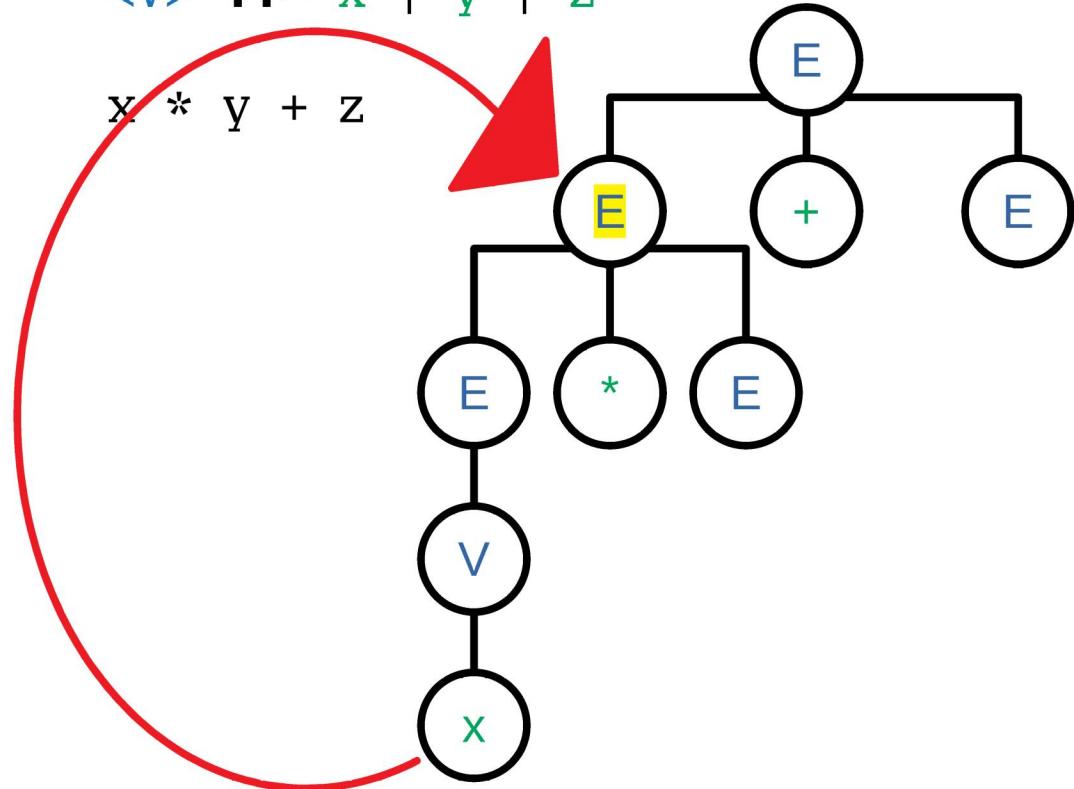
$\langle V \rangle ::= x \mid y \mid z$

x * y + z



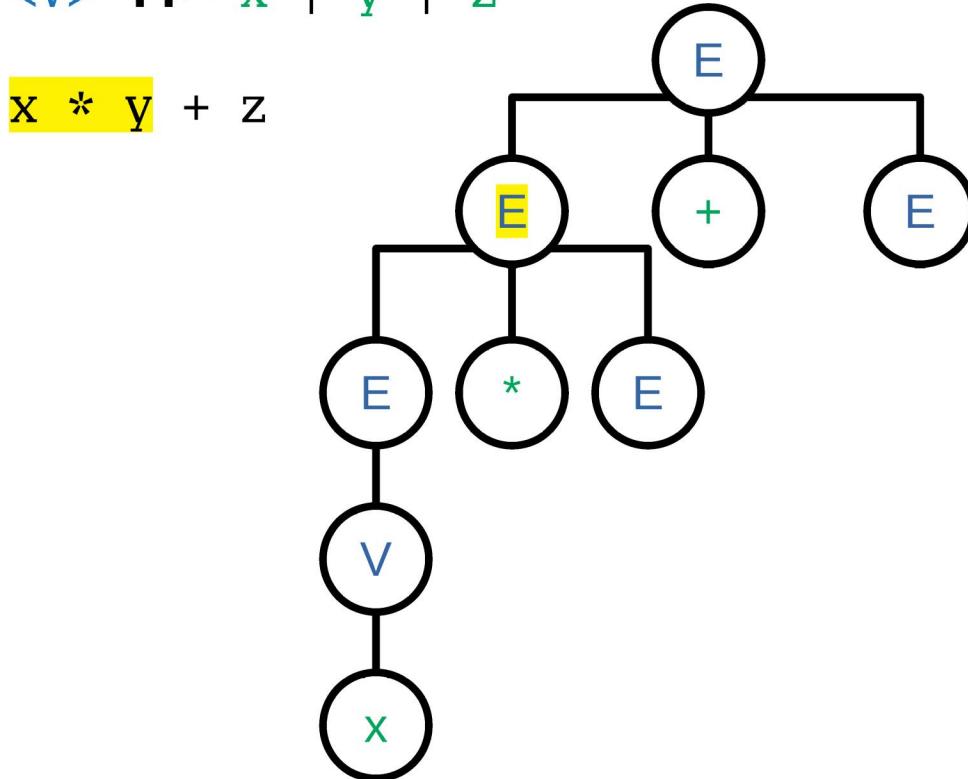
Parseträd – Bygg träd

$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$
 $\langle V \rangle ::= x \mid y \mid z$



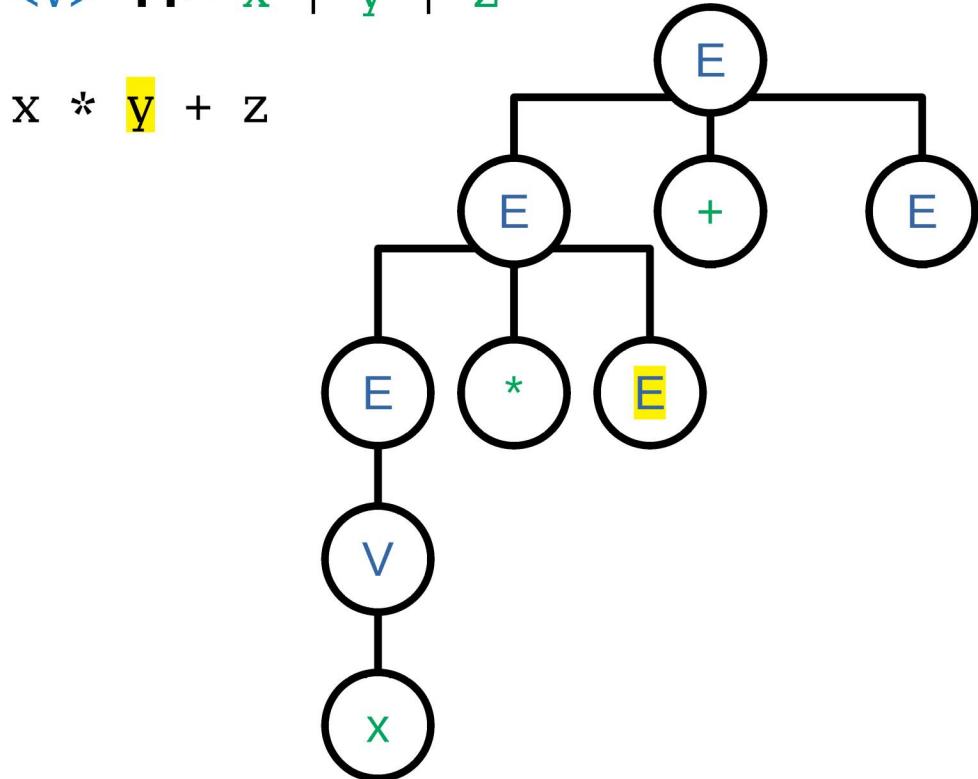
Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



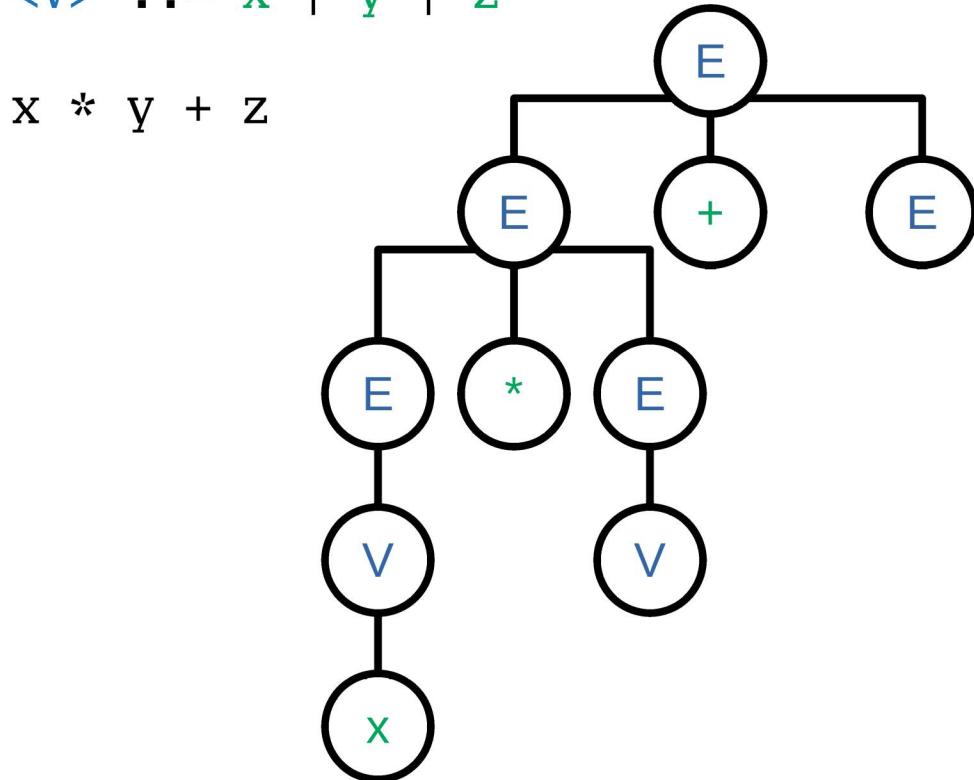
Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



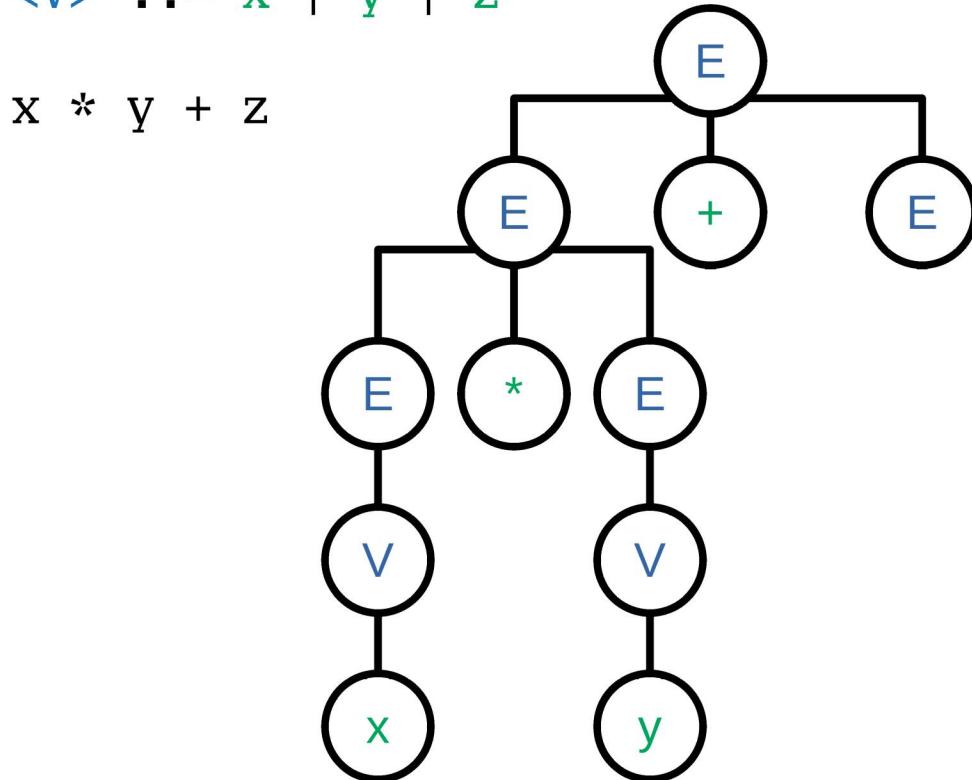
Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



Parseträd – Bygg träd

```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```

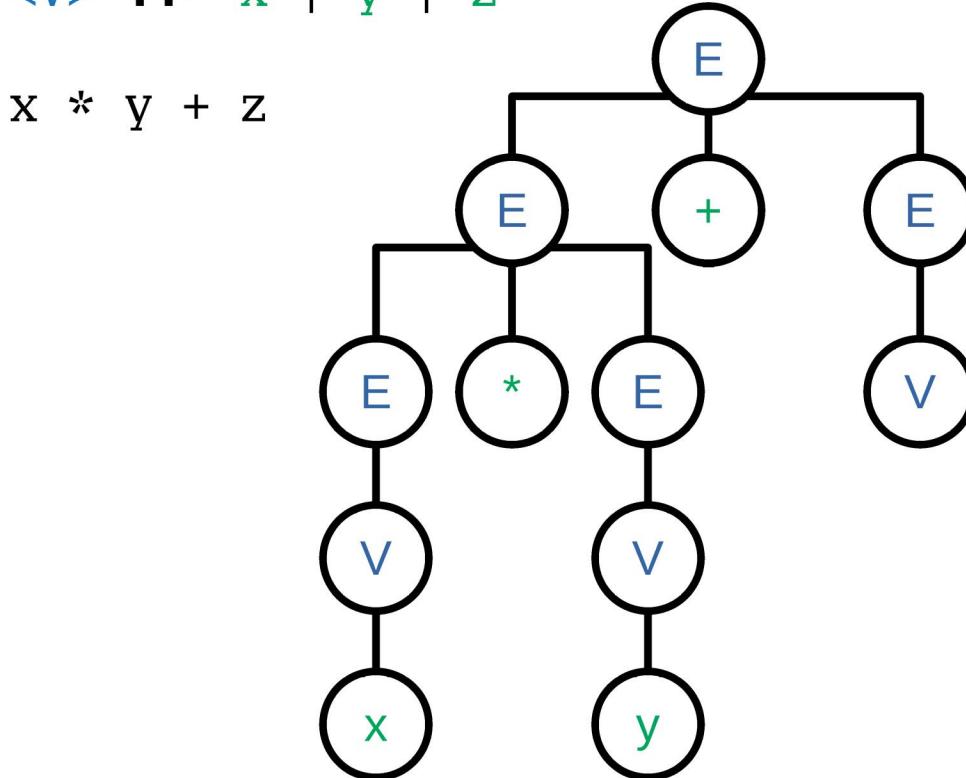


Parseträd – Bygg träd

67

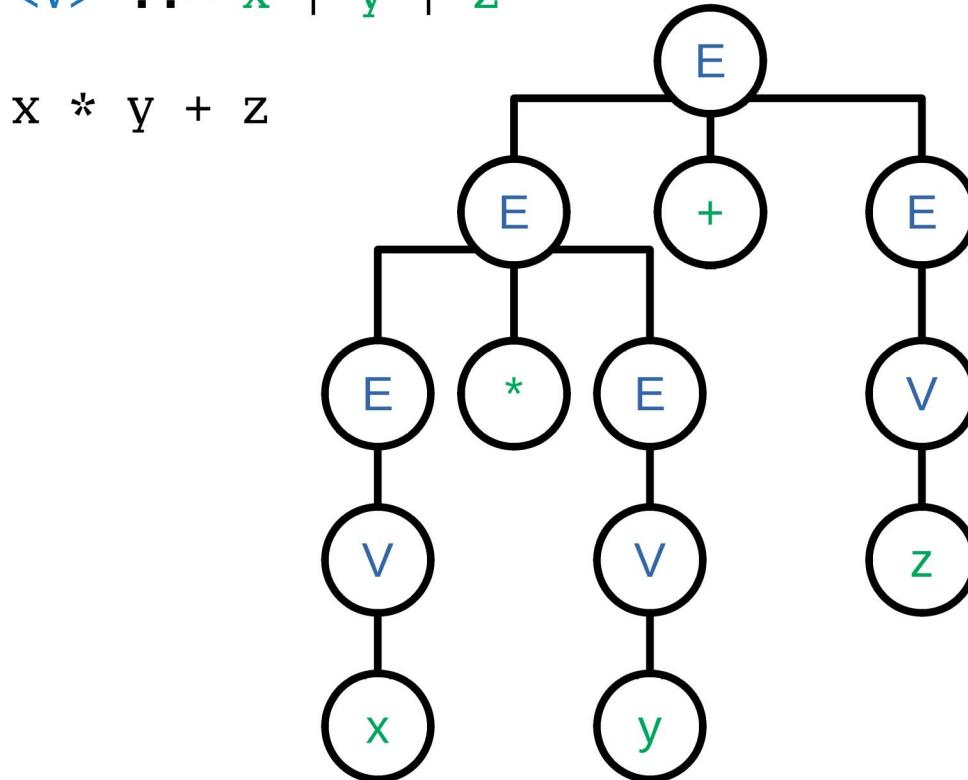
$\langle E \rangle ::= \langle E \rangle + \langle E \rangle \mid \langle E \rangle * \langle E \rangle \mid \langle V \rangle$

$\langle V \rangle ::= x \mid y \mid z$



Parseträd – Bygg träd

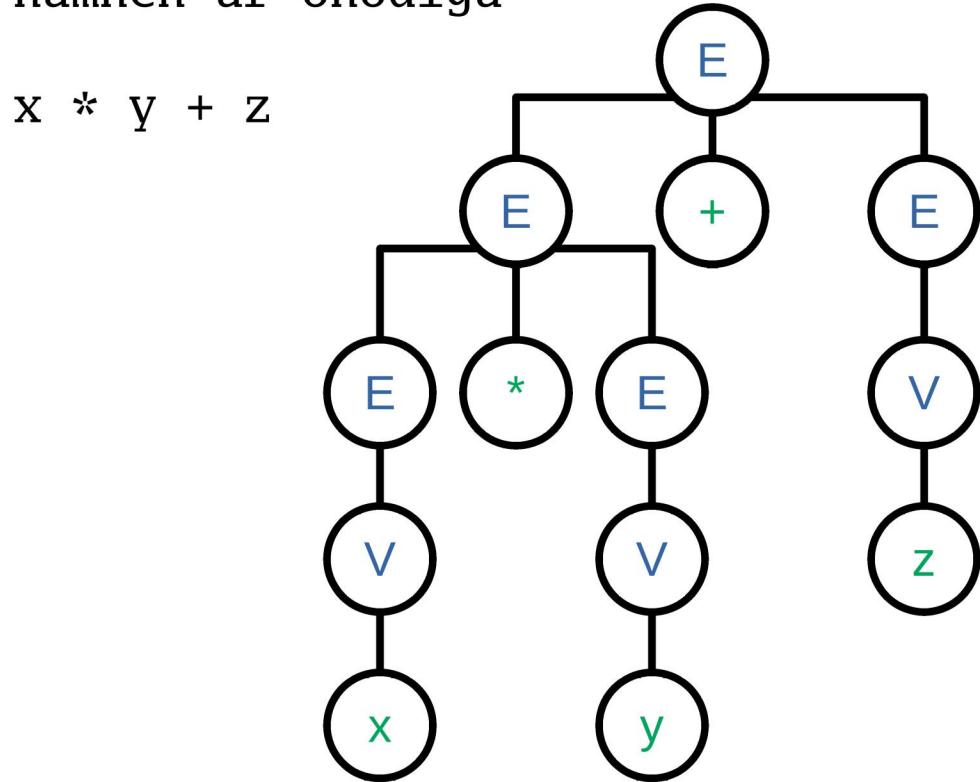
```
<E> ::= <E> + <E> | <E> * <E> | <V>  
<V> ::= x | y | z
```



Abstrakta syntaxträd - AST

69

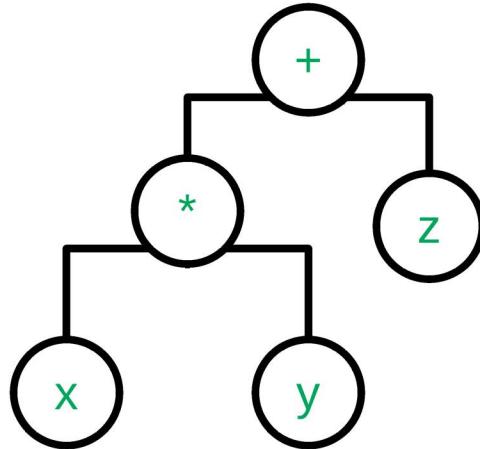
Detta byggs inte egentligen av datorn,
namnen är onödiga



Abstrakta syntaxträd - AST

Detta är närmare till vad datorn faktiskt kommer skapa

x * y + z

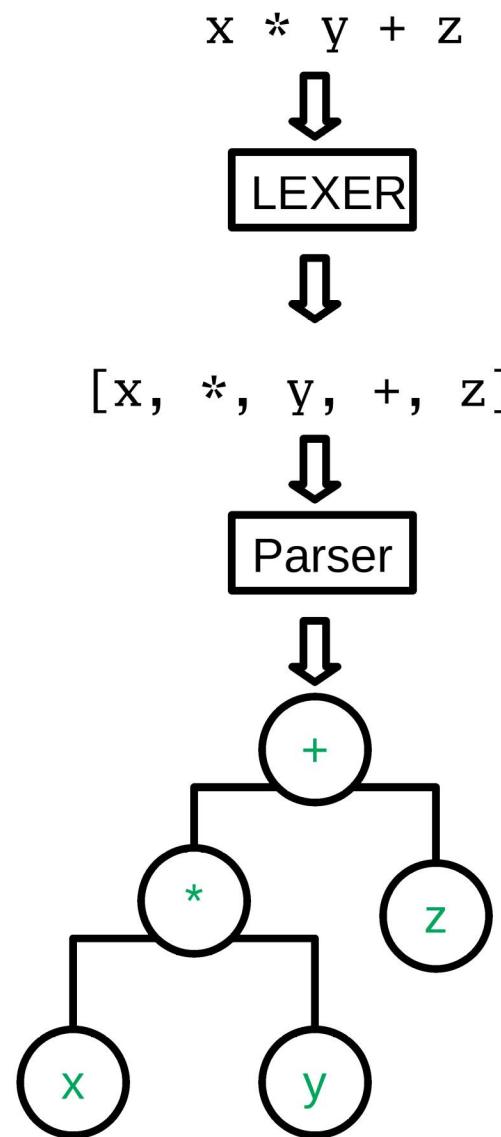


Parseträd vs AST

- Parseträg ligger närmare grammatiken
- AST är vad som genereras vid körning

Parser

72



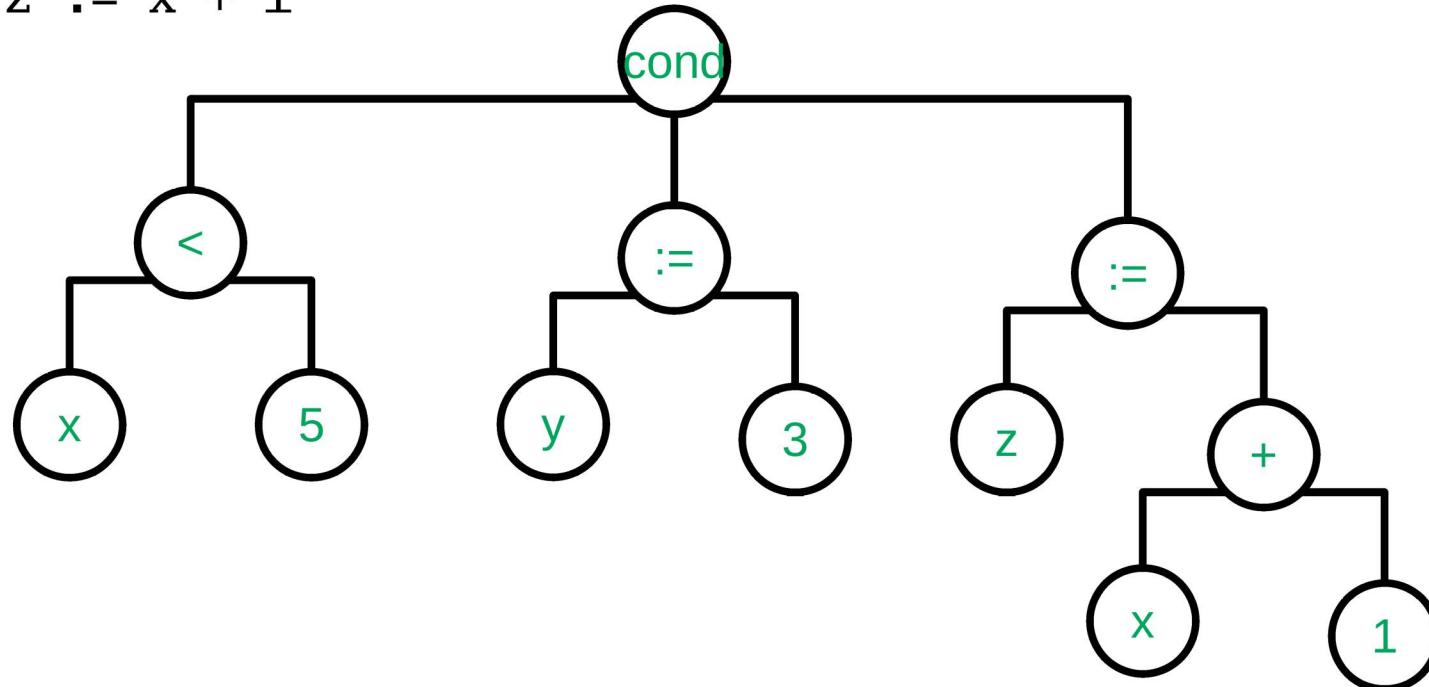
Ett till exempel

if x < 5 then

 y := 3

else then

 z := x + 1



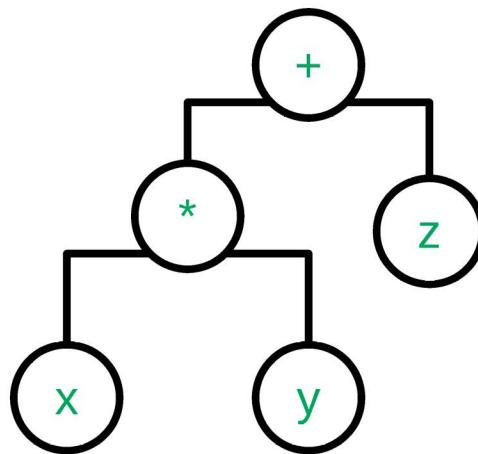
Resultat av parsning (AST)

Vad skapas?

75

Om detta är vad datorn ska skapa...
Vad är egentligen det här trädet?

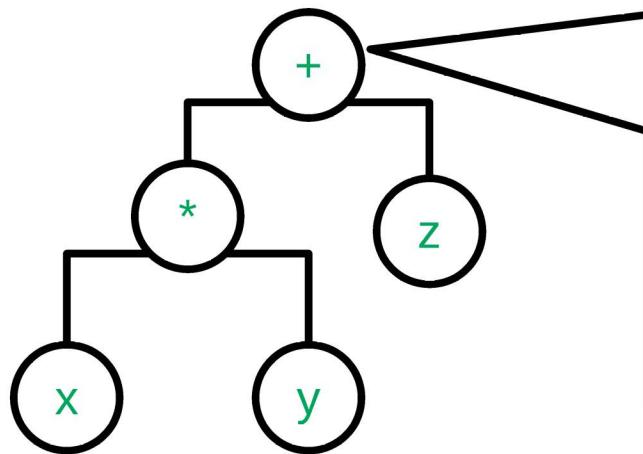
x * y + z



Vad skapas?

Om detta är vad datorn ska skapa...
Vad är egentligen det här trädet?

$x * y + z$



```
class Addition
  def initialize(lhs, rhs)
    @lhs = lhs
    @rhs = rhs
  end

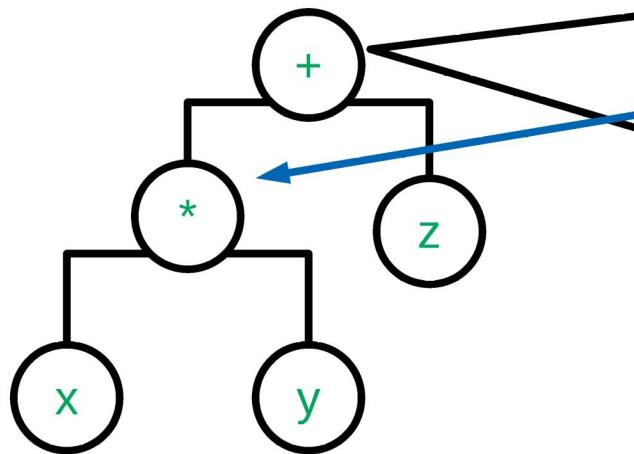
  def evaluate
    @lhs.evaluate + @rhs.evaluate
  end
end
```

Vad skapas?

77

Om detta är vad datorn ska skapa...
Vad är egentligen det här trädet?

$x * y + z$



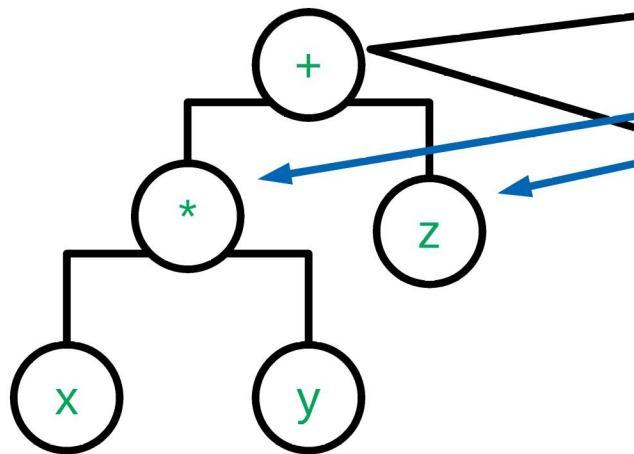
```
class Addition  
def initialize(lhs, rhs)  
  @lhs = lhs  
  @rhs = rhs  
end  
  
def evaluate  
  @lhs.evaluate + @rhs.evaluate  
end  
end
```

Vad skapas?

78

Om detta är vad datorn ska skapa...
Vad är egentligen det här trädet?

$x * y + z$



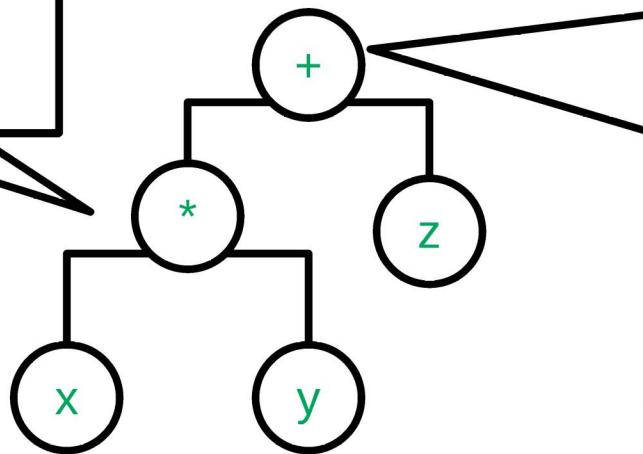
```
class Addition
  def initialize(lhs, rhs)
    @lhs = lhs
    @rhs = rhs
  end

  def evaluate
    @lhs.evaluate + @rhs.evaluate
  end
end
```

Vad skapas?

Om detta är vad datorn ska skapa...
 Vad är egentligen det här trädet?

```
class Multiplication
  def initialize(lhs, rhs)
    ...
  def evaluate
    ...
  end
```



```
class Addition
  def initialize(lhs, rhs)
    @lhs = lhs
    @rhs = rhs
  end

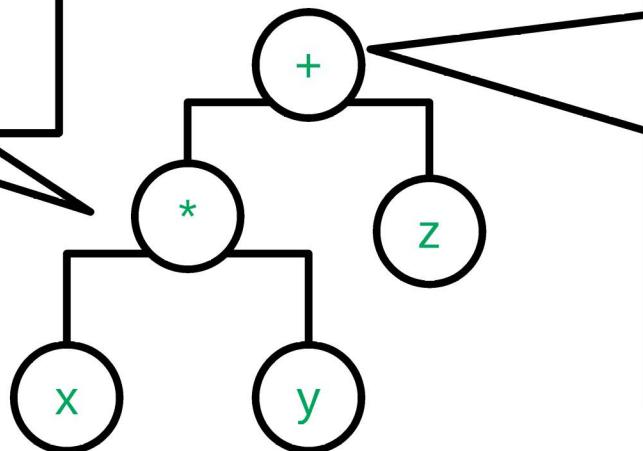
  def evaluate
    @lhs.evaluate + @rhs.evaluate
  end
end
```

Vad skapas?

80

Om detta är vad datorn ska skapa...
Vad är egentligen det här trädet?

```
class Multiplication  
def initialize(lhs, rhs)  
...  
def evaluate  
...
```



```
class Addition  
def initialize(lhs, rhs)  
@lhs = lhs  
@rhs = rhs  
end  
  
def evaluate  
@lhs.evaluate + @rhs.evaluate  
end  
end
```

```
class Variable  
def initialize(name)  
...  
def evaluate  
return VARS[@name]  
end
```

Hur?

- Vi vill att vår parser genererar detta träd i form av objekt, men hur åstadkommer vi detta?

Hur?

- Vi vill att vår parser genererar detta träd i form av objekt, men hur åstadkommer vi detta?

```
match(:term, '*', :dice) { |a, _, b| a * b }
```

Hur?

- Vi vill att vår parser genererar detta träd i form av objekt, men hur åstadkommer vi detta?

```
match(:term, '*', :dice) { |a, _, b| a * b }
```

```
match(:term, '*', :dice) { |a, _, b| Multiplication.new(a, b) }
```

Hur?

- Vi vill att vår parser genererar detta träd i form av objekt, men hur åstadkommer vi detta?

```
match(:term, '*', :dice) { |a, _, b| a * b }
```

```
match(:term, '*', :dice) { |a, _, b| Multiplication.new(a, b) }
```

Fullständig AST-lösning för
DiceRoller från FÖ i TDP007

Grammatiska problem

Funktionsanrop

- Vi vill att alla dessa fungerar:

foo(x,y,z)

foo(x)

foo()

Funktionsanrop

foo(x,y,z)
foo(x)
foo()

<call> ::= <name> (<params>)

<params> ::= <params>, <param>
| <param>
| <empty>

Funktionsanrop

foo(x,y,z)
foo(x)
foo()

<call> ::= <name> (<params>)

<params> ::= <params>, <param>
| <param>
| <empty>



Funktionsanrop

foo(x,y,z)
foo(x)
foo()

<call> ::= <name> (<params>)

<params> ::= <params>, <param>
| <param>
| <empty>

foo(,x)



Hur blir detta fel?

Funktionsanrop en lösning

```
foo(x,y,z)
foo(x)
foo
```

```
<call> ::= <name> (<params>)
          | <name> () |
```

```
<params> ::= <params>, <param>
           | <param>
```

www.liu.se