

5.3 Auswertung von Ausdrücken

Funktionen in **Java** bekommen **Parameter**/Argumente als Input, und liefern als Output den Wert eines vorbestimmten Typs. Zum Beispiel könnte man eine Funktion

```
int min(int a, int b)
```

implementieren, die das Minimum ihrer Argumente zurückliefert.

Operatoren sind spezielle vordefinierte Funktionen, die in **Infix**-Notation geschrieben werden (wenn sie binär sind):

```
a + b = +(a,b)
```

5.3 Auswertung von Ausdrücken

Ein **Ausdruck** ist eine Kombination von Literalen, Operatoren, Funktionen, Variablen und Klammern, die verwendet wird, um einen Wert zu berechnen.

Beispiele: (x z.B. vom Typ `int`)

- ▶ $7 + 4$
- ▶ $3 / 5 + 3$
- ▶ $\text{min}(3, x) + 20$
- ▶ $x = 7$
- ▶ $x *= 2$

Unäre Operatoren:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
++	Post-inkrement	(var) Zahl, char	keine	2
--	Post-dekrement	(var) Zahl, char	keine	2
++	Pre-inkrement	(var) Zahl, char	rechts	3
--	Pre-dekrement	(var) Zahl, char	rechts	3
+	unäres Plus	Zahl, char	rechts	3
-	unäres Minus	Zahl, char	rechts	3
!	Negation	boolean	rechts	3

Achtung



Prefix- und Postfixoperator

- ▶ Die Operatoranwendungen `++x` und `x++` inkrementieren beide den Wert der Variablen `x` (als **Seiteneffekt**).
- ▶ `++x` tut das, **bevor** der Wert des Ausdrucks ermittelt wird (**Pre-Inkrement**).
- ▶ `x++` tut das, **nachdem** der Wert ermittelt wurde (**Post-Inkrement**).

- ▶ `b = x++;` entspricht:

```
b = x;  
x = x + 1;
```

- ▶ `b = ++x;` entspricht:

```
x = x + 1;  
b = x;
```

Operatoren

Binäre arithmetische Operatoren:

byte, short, char werden nach int konvertiert

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
*	Multiplikation	Zahl, char	links	4
/	Division	Zahl, char	links	4
%	Modulo	Zahl, char	links	4
+	Addition	Zahl, char	links	5
-	Subtraktion	Zahl, char	links	5

Konkatenation

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
+	Konkatenation	String	links	5

Vergleichsoperatoren:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
>	größer	Zahl, char	keine	7
>=	größergleich	Zahl, char	keine	7
<	kleiner	Zahl, char	keine	7
<=	kleinergleich	Zahl, char	keine	7
==	gleich	alle	links	8
!=	ungleich	alle	links	8

Boolsche Operatoren:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
&&	Und-Bedingung	boolean	links	12
	Oder-Bedingung	boolean	links	13

Zuweisungsoperatoren:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
=	Zuweisung	(links var) alle	rechts	15
*=, /=, %= +=, -=	Zuweisung	(links var) alle	rechts	15

Für die letzte Form gilt:

$$v \Leftarrow a \iff v = (\text{type}(v)) (v \circ a)$$

Warnung:

- ▶ Eine Zuweisung $x = y$; ist in Wahrheit ein **Ausdruck**.
- ▶ Der Wert ist der Wert der rechten Seite.
- ▶ Die Modifizierung der Variablen x erfolgt als **Seiteneffekt**.
- ▶ Das Semikolon ';' hinter einem Ausdruck wirft nur den Wert weg.

Fatal für Fehler in Bedingungen:

```
boolean x = false;  
if (x = true)  
    write("Sorry! This must be an error ...");
```

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
- ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
- ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
- ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
- ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
- ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
- ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
 - ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
 - ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
 - ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
- ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
- ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
- ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
- ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
- ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
- ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Assoziativität

- ▶ Die Assoziativität entscheidet über die Reihenfolge bei Operatoren gleicher Priorität. (links = der linkeste Operator wird zuerst ausgeführt)
- ▶ Alle Operatoren einer Prioritätsgruppe haben dieselbe Assoziativität.
- ▶ Bis auf Zuweisungsoperatoren (=, +=, etc.) sind alle binären Operatoren linksassoziativ.
- ▶ unäre Operatoren, die ihr Argument rechts erwarten sind rechtsassoziativ
- ▶ unäre Operatoren, die ihr Argument links erwarten (postfix-Operatoren ++, --) sind linksassoziativ
- ▶ Der ternäre Bedingungsoperator (später) ist rechtsassoziativ

5.3 Auswertung von Ausdrücken

Die Auswertung eines Ausdrucks liefert

- ▶ eine Variable (**var**),
- ▶ einen reinen Wert (**val**) oder
- ▶ void (**void**)

In den ersten beiden Fällen hat der Ausdruck dann einen

- ▶ Typ, z.B.: **int**, und einen
- ▶ Wert, z.B.: **42**

Für z.B. Zuweisungen muss die Auswertung des Ausdrucks auf der linken Seite eine Variable ergeben!!!

5.3 Auswertung von Ausdrücken

In **Java** werden Unterausdrücke von links nach rechts ausgewertet. D.h. um den Wert einer Operation zu berechnen:

- ▶ werte (rekursiv) alle Operanden von links nach rechts aus
- ▶ führe die Operation auf den Resultaten aus

Ausnahmen: `||`, `&&`, und der ternäre Bedingungsoperator `?:`, werten nicht alle Operanden aus (**Kurzschlussauswertung**).

Man sollte nie Ausdrücke formulieren, deren Ergebnis von der Auswertungsreihenfolge abhängt!!!

5.3 Auswertung von Ausdrücken

In **Java** werden Unterausdrücke von links nach rechts ausgewertet. D.h. um den Wert einer Operation zu berechnen:

- ▶ werte (rekursiv) alle Operanden von links nach rechts aus
- ▶ führe die Operation auf den Resultaten aus

Ausnahmen: `||`, `&&`, und der ternäre Bedingungsoperator `?:`, werten nicht alle Operanden aus (**Kurzschlussauswertung**).

Man sollte nie Ausdrücke formulieren, deren Ergebnis von der Auswertungsreihenfolge abhängt!!!

5.3 Auswertung von Ausdrücken

Im Folgenden betrachten wir Klammern als einen Operator der nichts tut:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
()	Klammerung	alle	links	0

Beispiel: $2 + x * (z - d)$

2 + x * (z - d)

Beispiel: $2 + x * (z - d)$



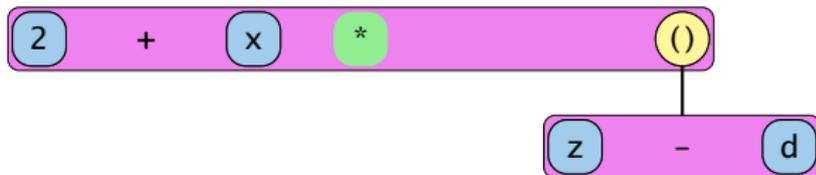
Beispiel: $2 + x * (z - d)$



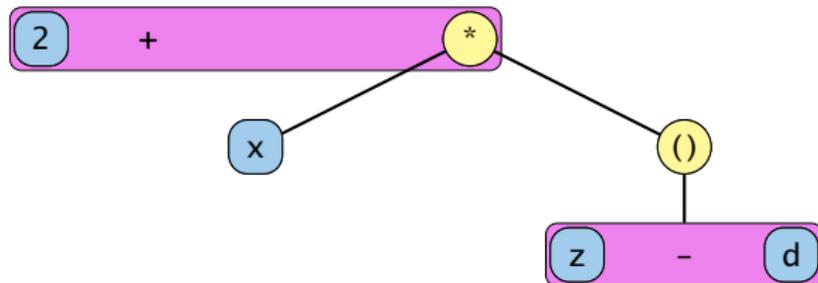
Beispiel: $2 + x * (z - d)$



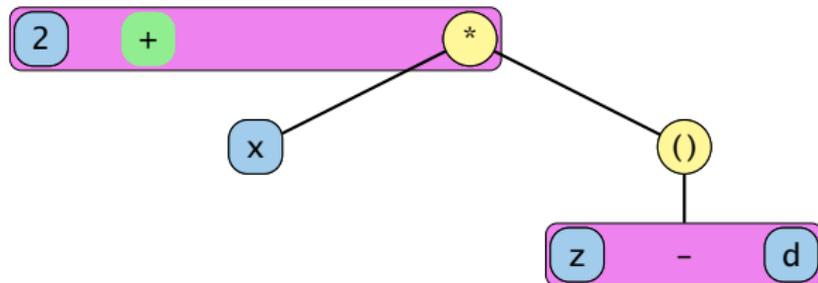
Beispiel: $2 + x * (z - d)$



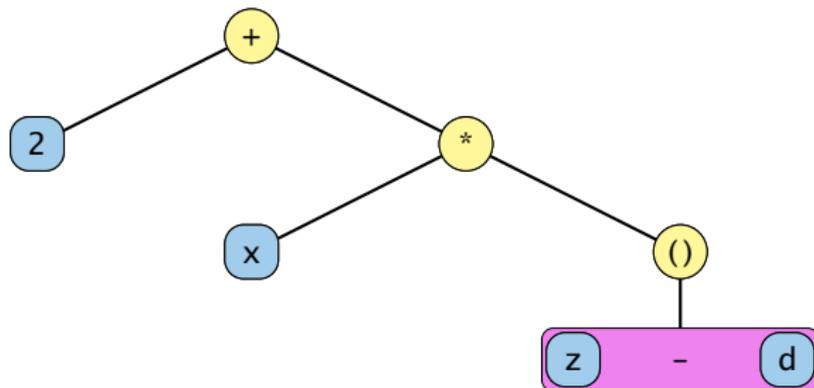
Beispiel: $2 + x * (z - d)$



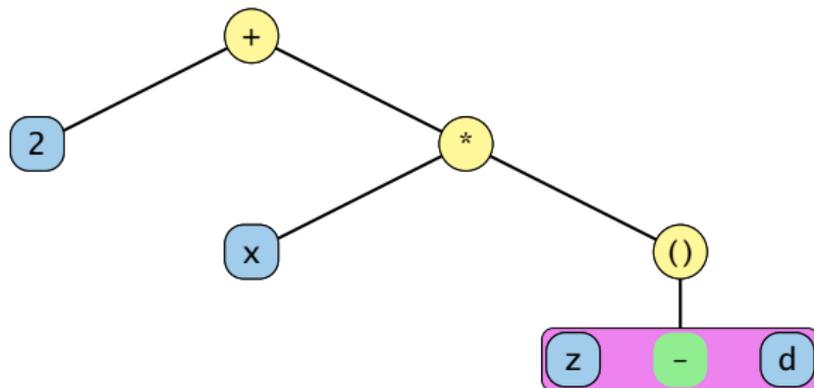
Beispiel: $2 + x * (z - d)$



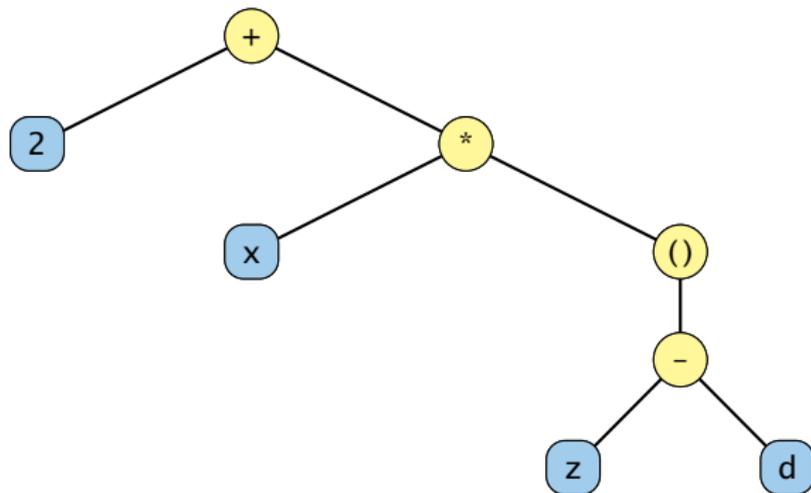
Beispiel: $2 + x * (z - d)$



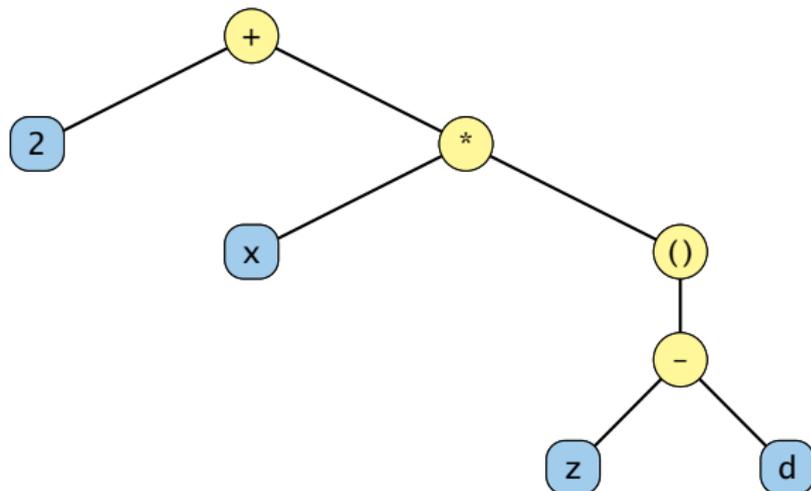
Beispiel: $2 + x * (z - d)$



Beispiel: $2 + x * (z - d)$



Beispiel: $2 + x * (z - d)$

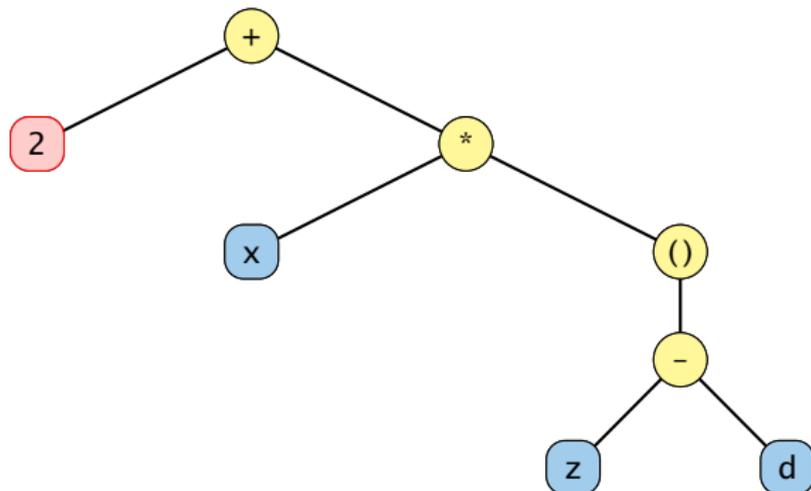


x

d

z

Beispiel: $2 + x * (z - d)$

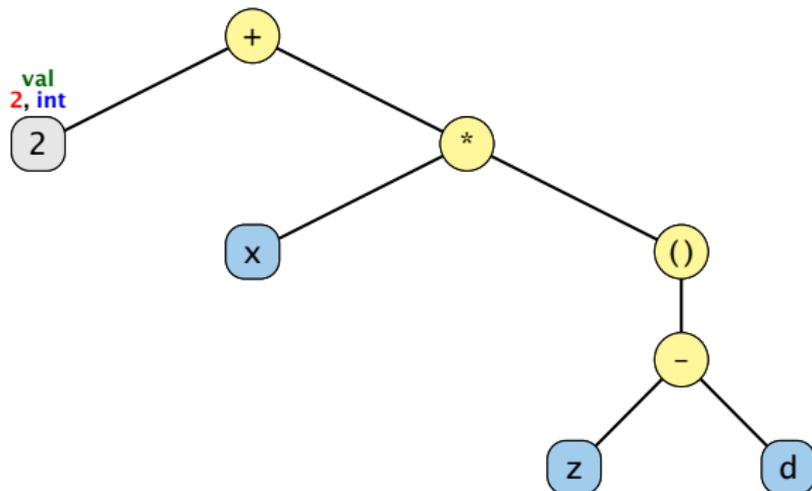


x

d

z

Beispiel: $2 + x * (z - d)$

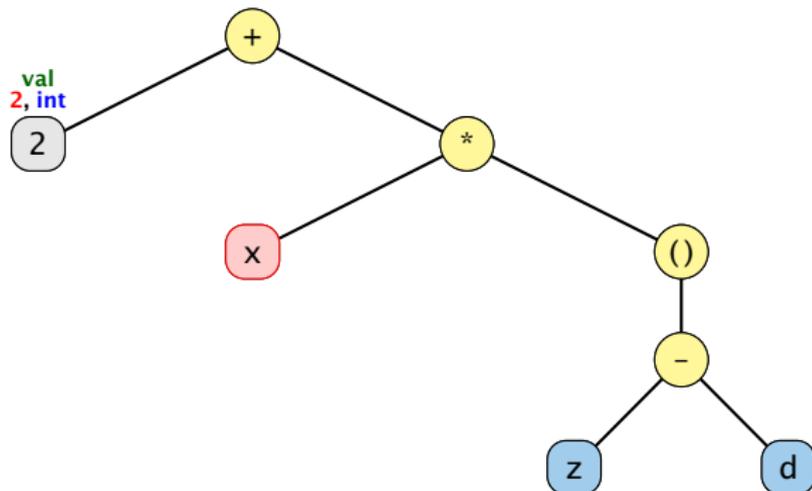


x

d

z

Beispiel: $2 + x * (z - d)$

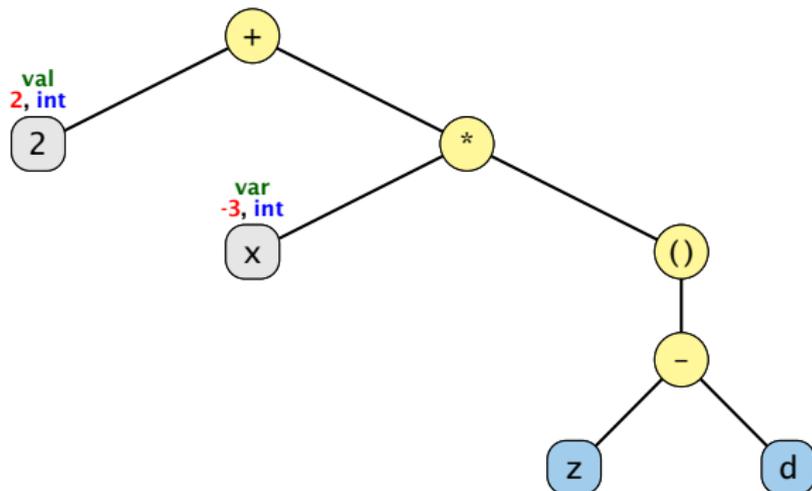


x

d

z

Beispiel: $2 + x * (z - d)$

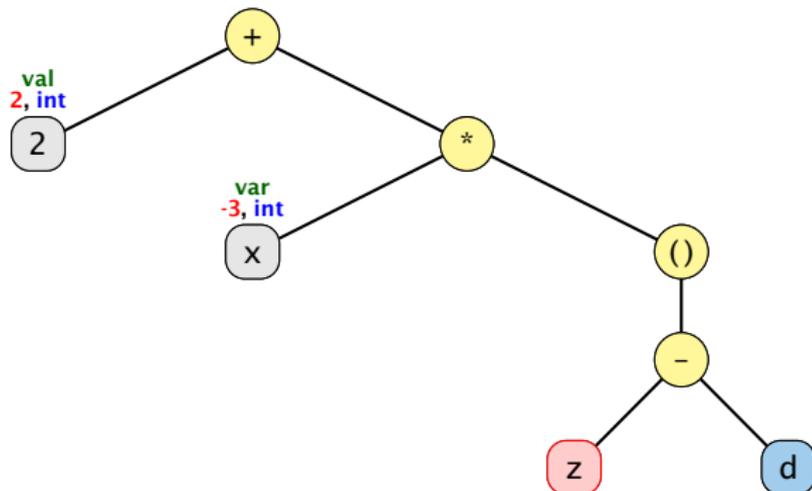


x

d

z

Beispiel: $2 + x * (z - d)$

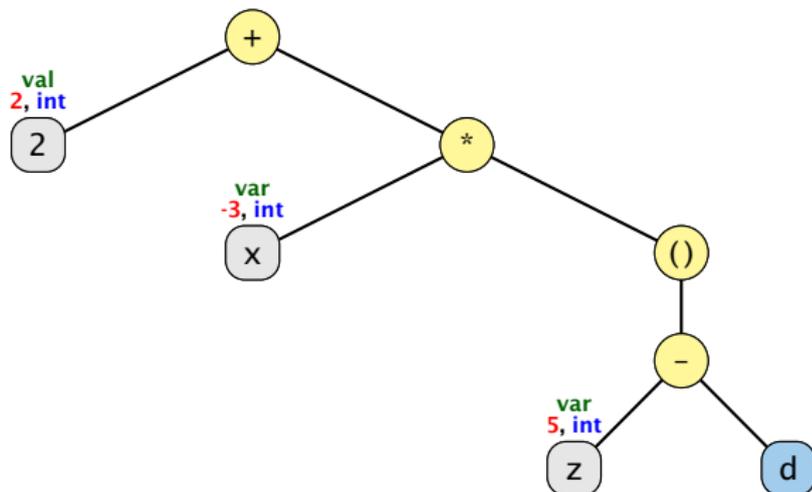


x

d

z

Beispiel: $2 + x * (z - d)$

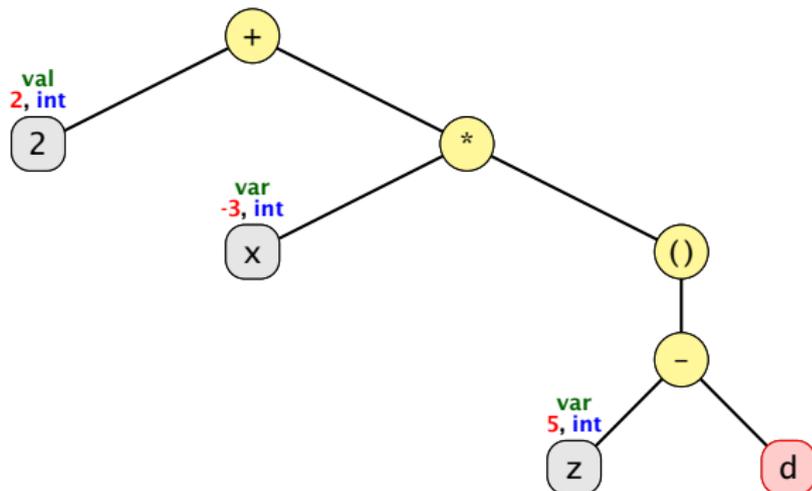


x

d

z

Beispiel: $2 + x * (z - d)$

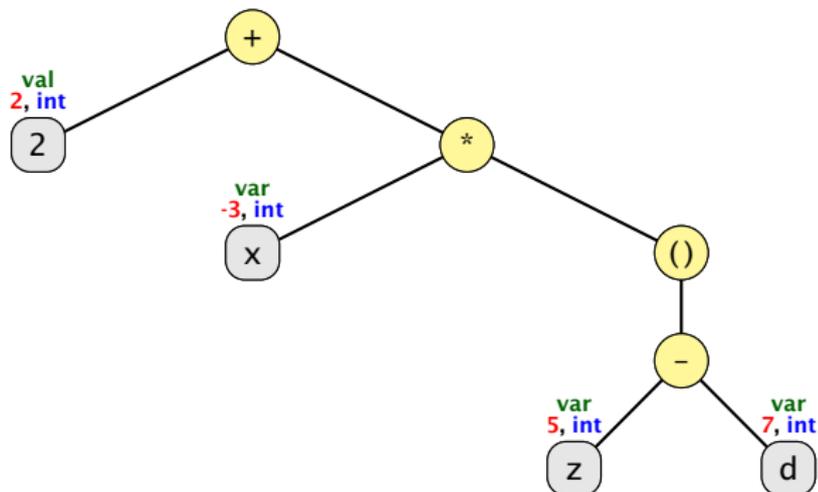


x

d

z

Beispiel: $2 + x * (z - d)$

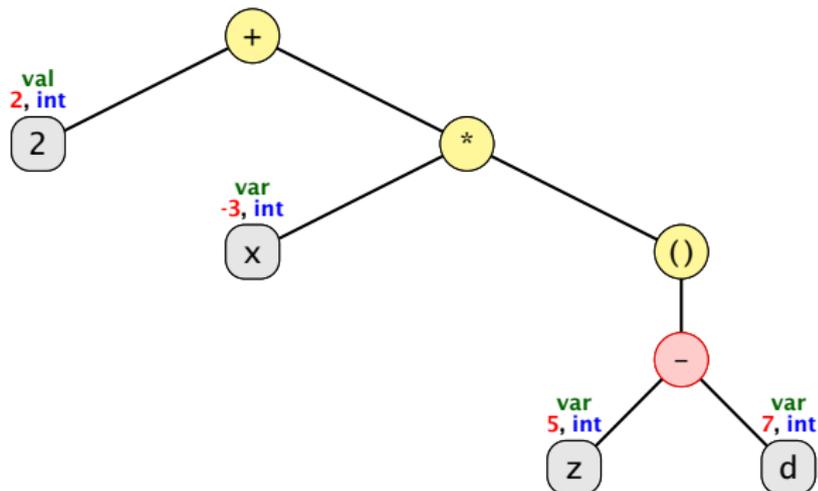


x

d

z

Beispiel: $2 + x * (z - d)$

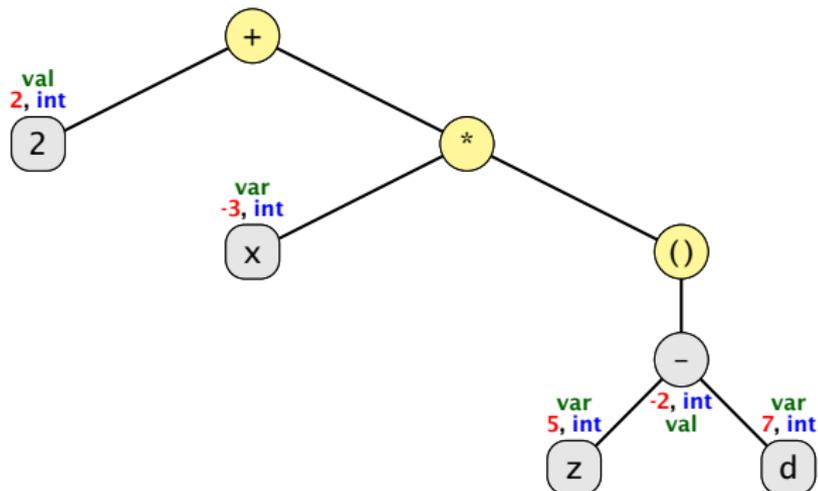


x

d

z

Beispiel: $2 + x * (z - d)$

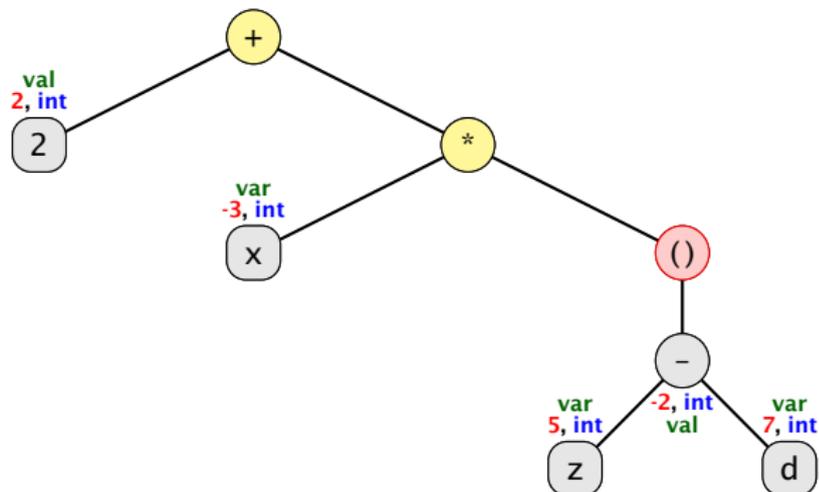


x

d

z

Beispiel: $2 + x * (z - d)$

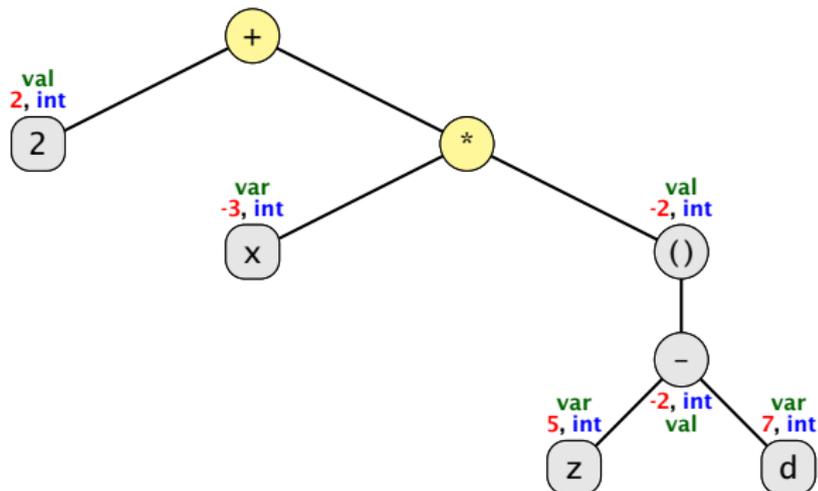


x

d

z

Beispiel: $2 + x * (z - d)$

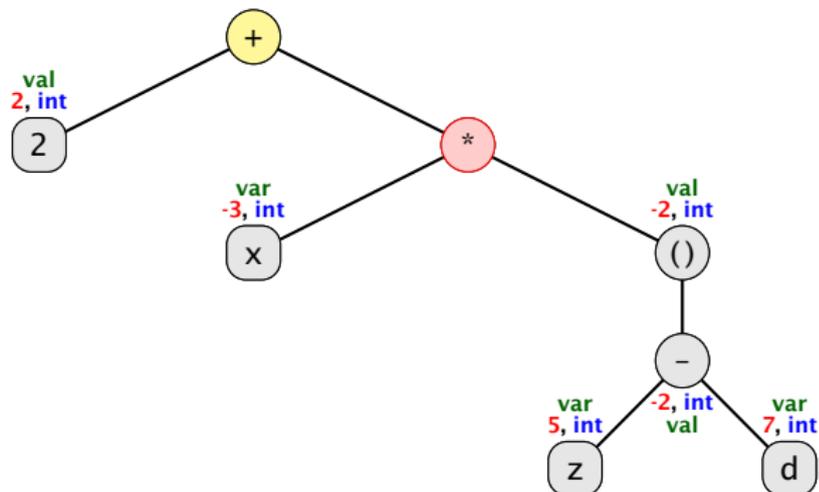


x

d

z

Beispiel: $2 + x * (z - d)$

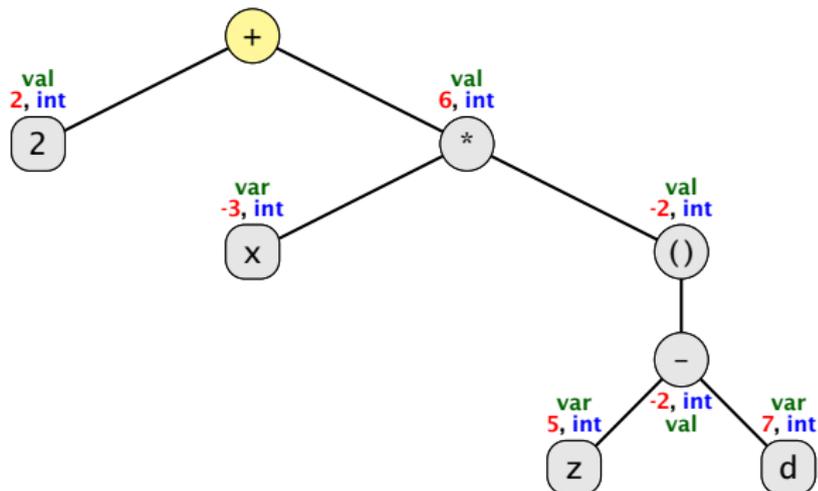


x

d

z

Beispiel: $2 + x * (z - d)$

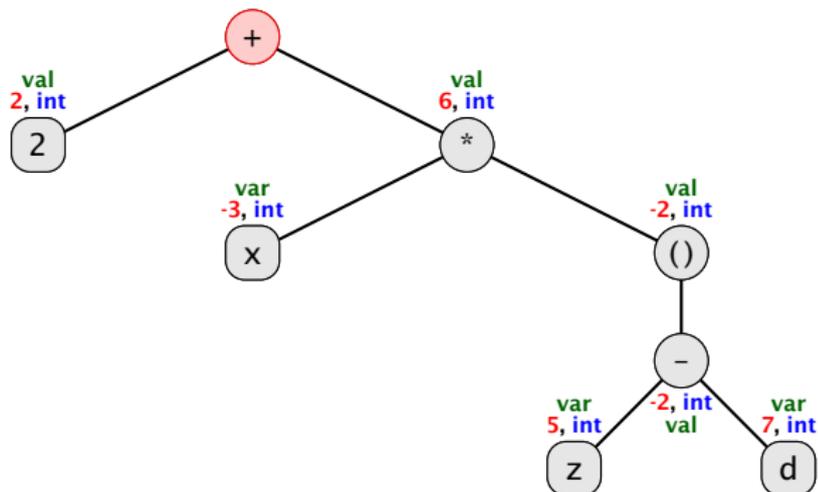


x

d

z

Beispiel: $2 + x * (z - d)$

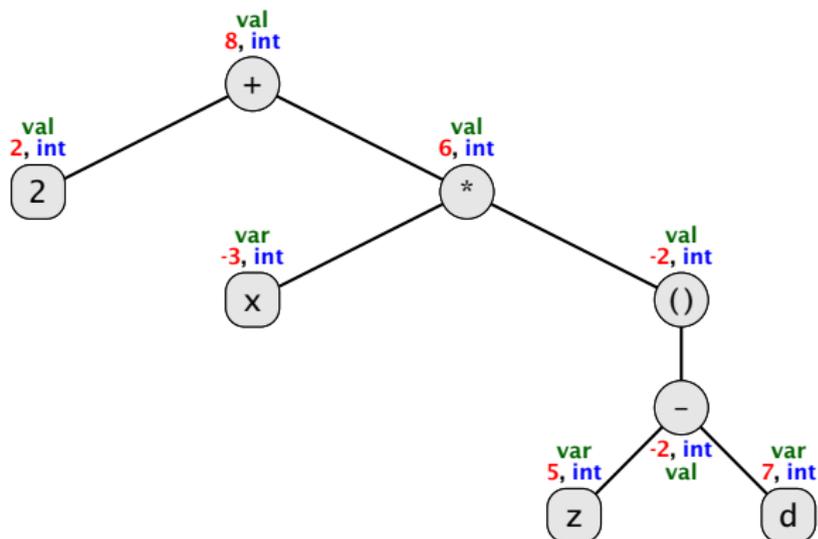


x

d

z

Beispiel: $2 + x * (z - d)$



x

d

z

Beispiel: $a = b = c = d = 0$

$a = b = c = d = 0$

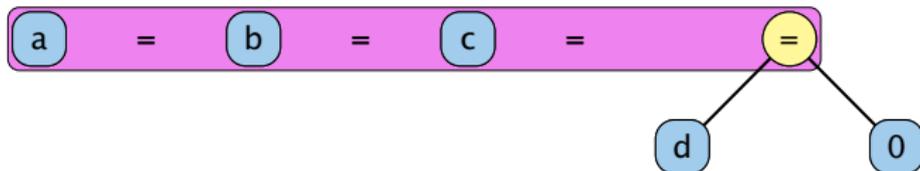
Beispiel: $a = b = c = d = 0$



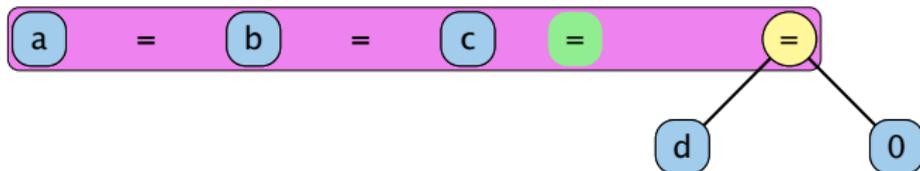
Beispiel: $a = b = c = d = 0$



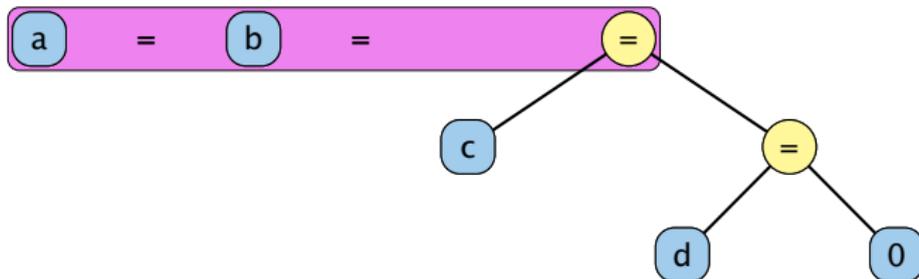
Beispiel: $a = b = c = d = 0$



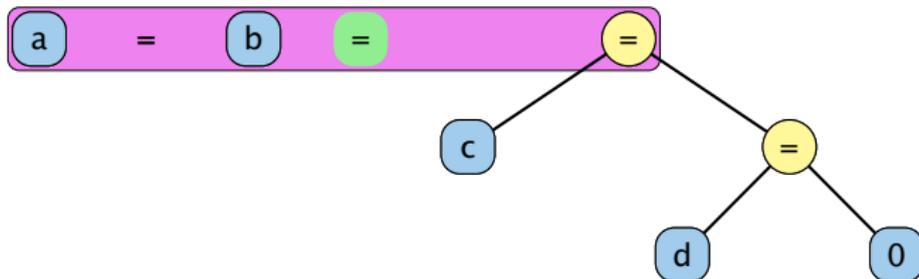
Beispiel: $a = b = c = d = 0$



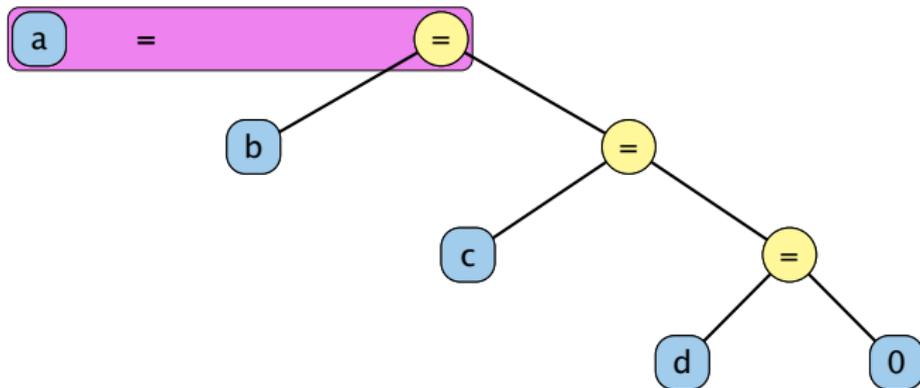
Beispiel: $a = b = c = d = 0$



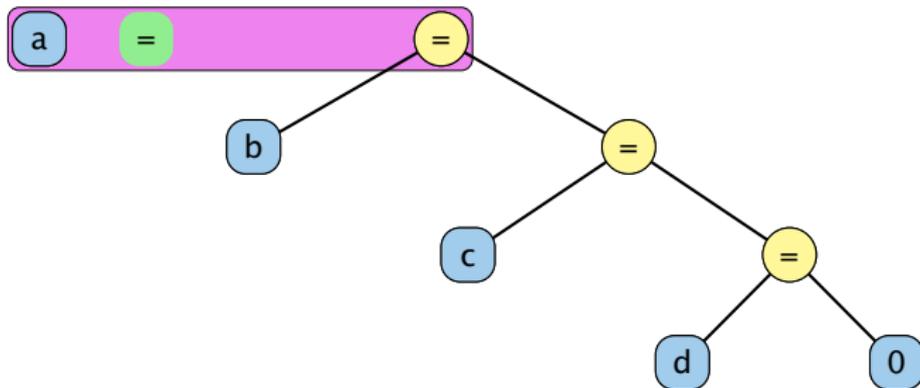
Beispiel: $a = b = c = d = 0$



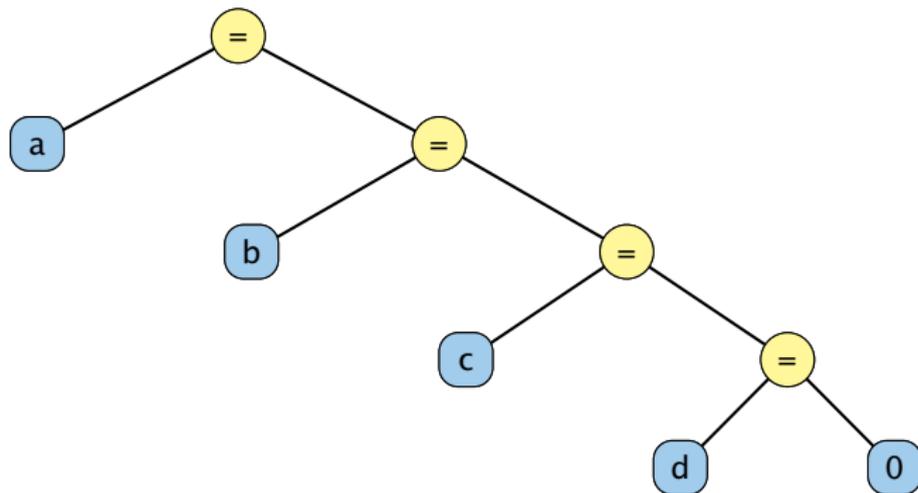
Beispiel: $a = b = c = d = 0$



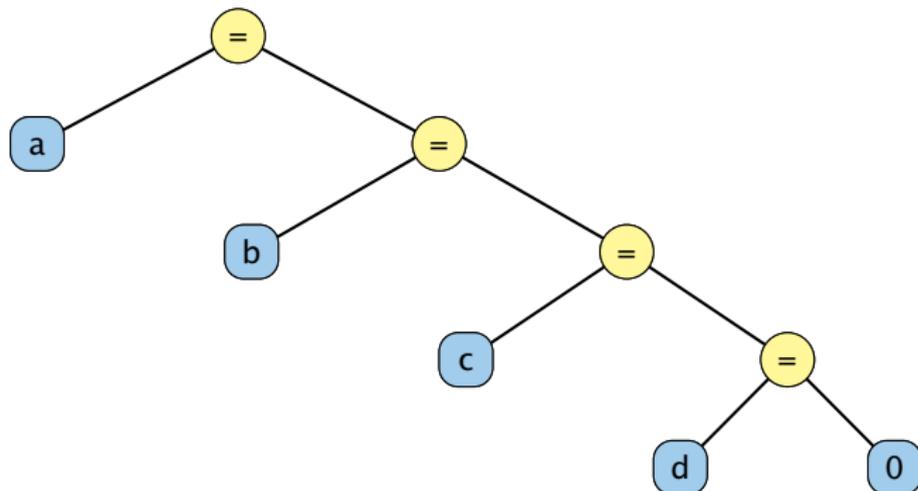
Beispiel: $a = b = c = d = 0$



Beispiel: $a = b = c = d = 0$



Beispiel: $a = b = c = d = 0$



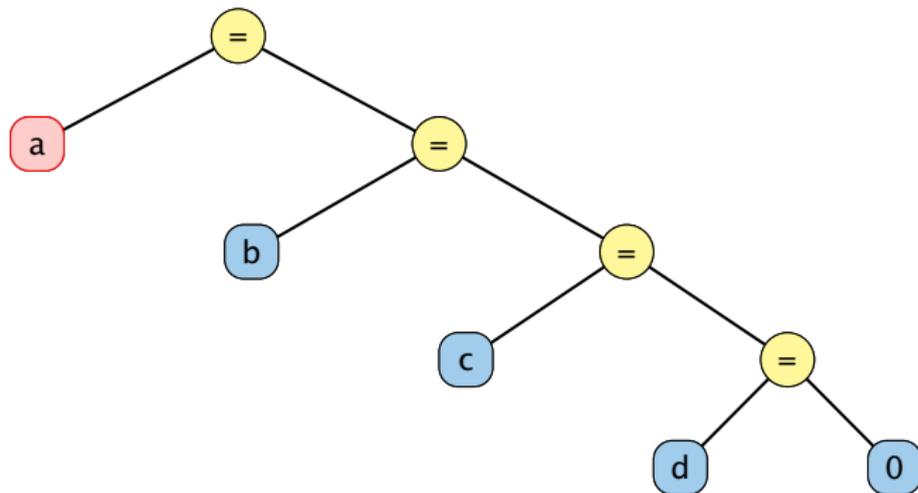
a

b

c

d

Beispiel: $a = b = c = d = 0$



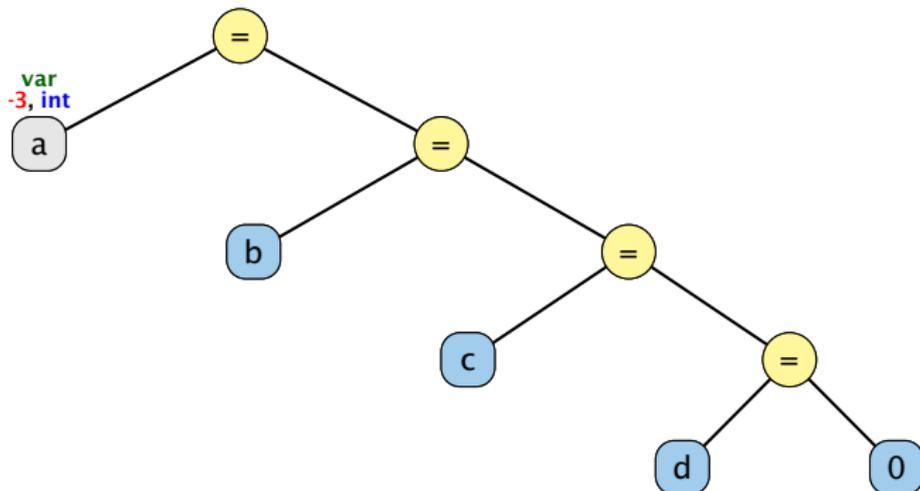
a

b

c

d

Beispiel: $a = b = c = d = 0$



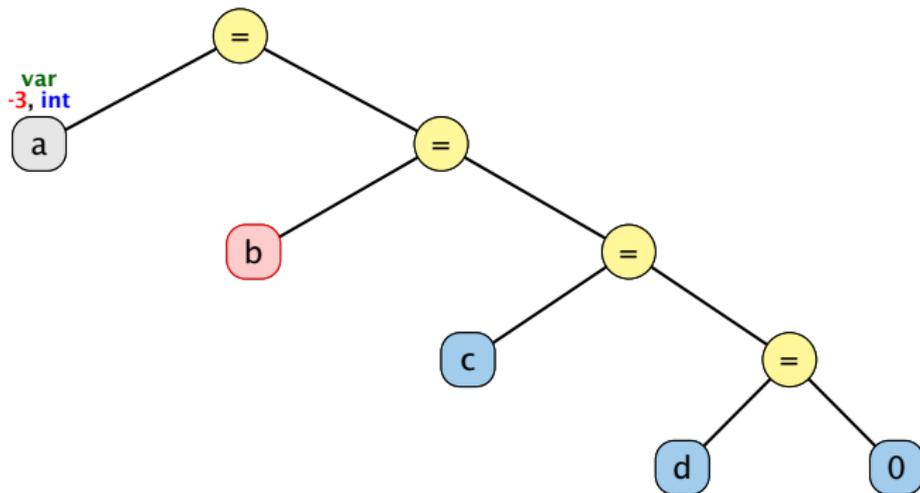
a

b

c

d

Beispiel: $a = b = c = d = 0$



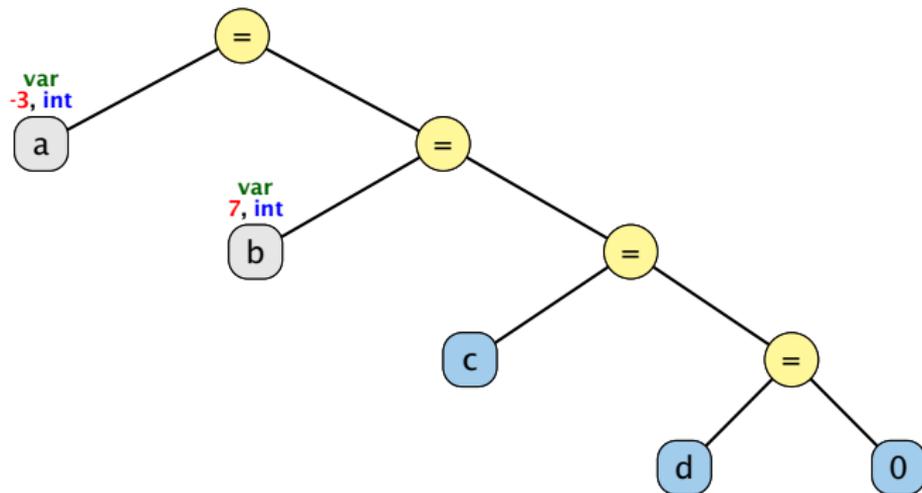
a

b

c

d

Beispiel: $a = b = c = d = 0$



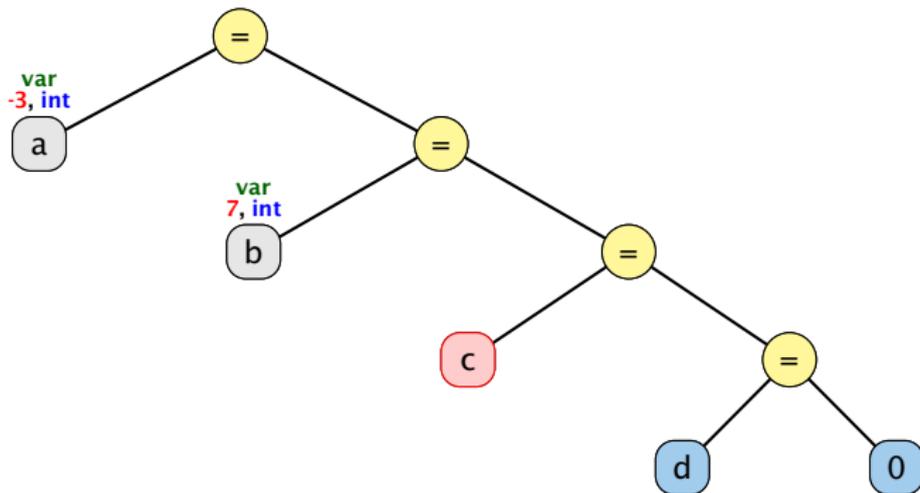
a

b

c

d

Beispiel: $a = b = c = d = 0$



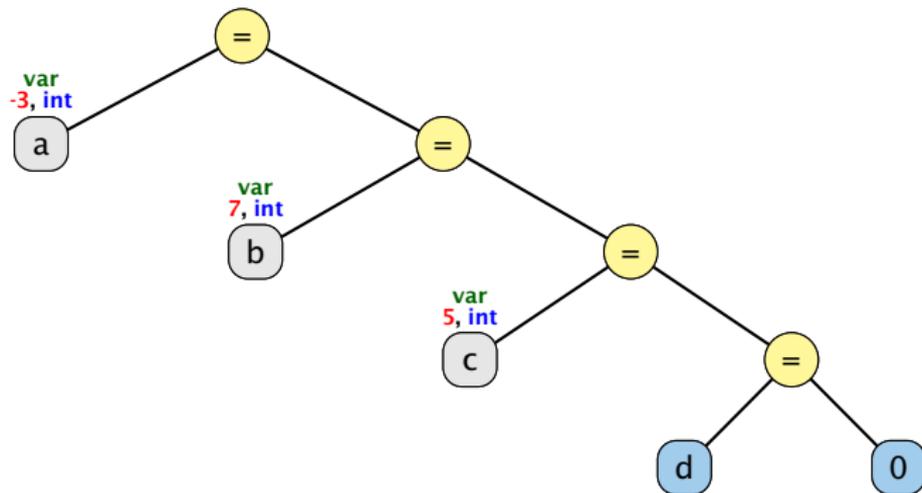
a

b

c

d

Beispiel: $a = b = c = d = 0$



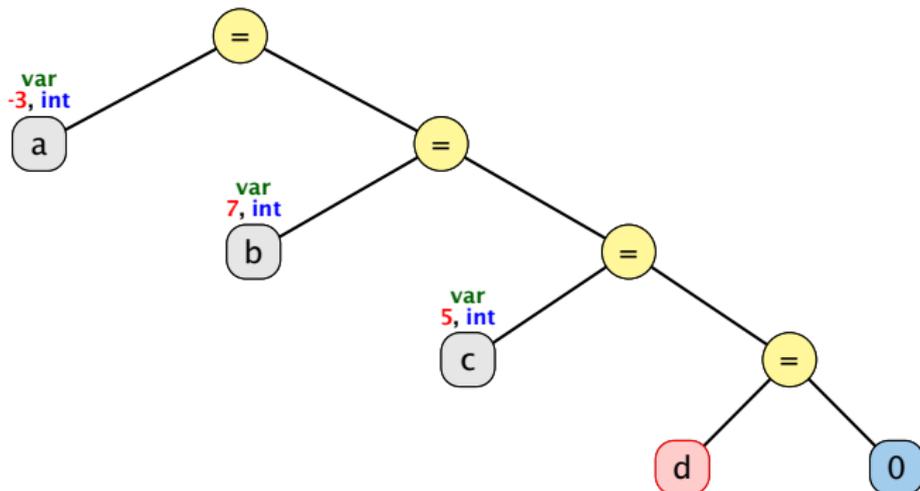
a

b

c

d

Beispiel: $a = b = c = d = 0$



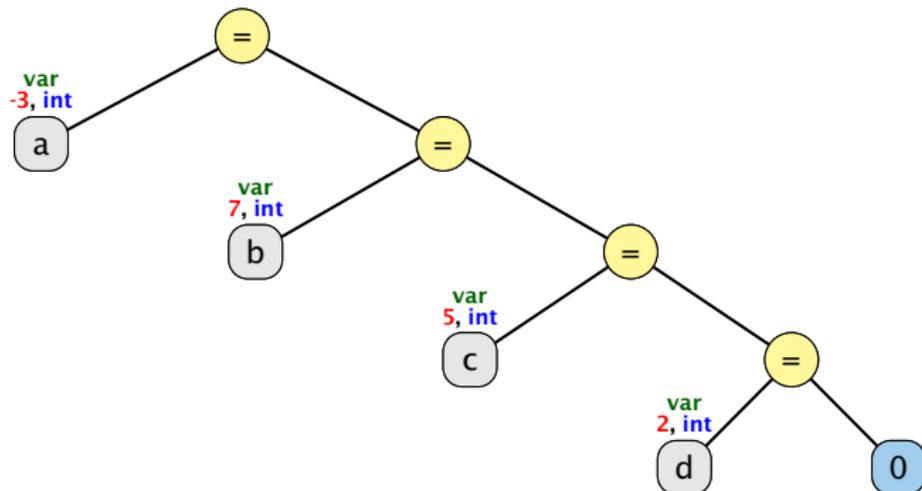
a [-3]

b [7]

c [5]

d [2]

Beispiel: $a = b = c = d = 0$



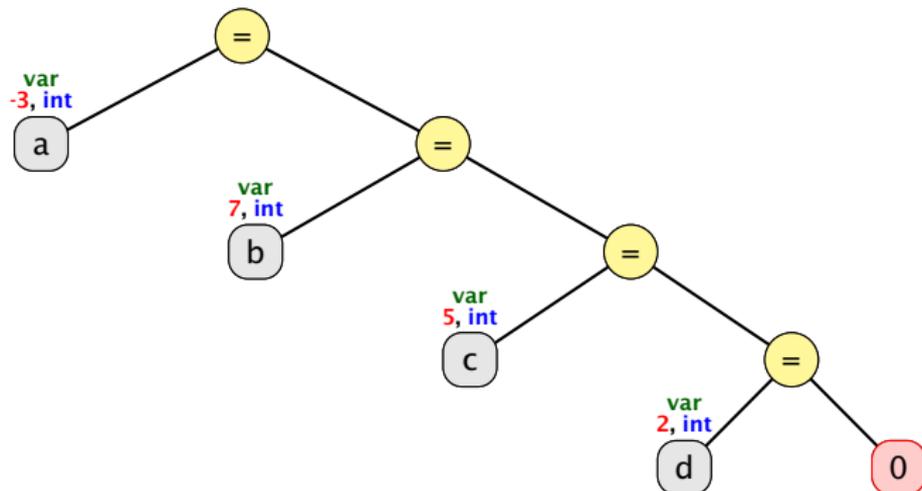
a [-3]

b [7]

c [5]

d [2]

Beispiel: $a = b = c = d = 0$



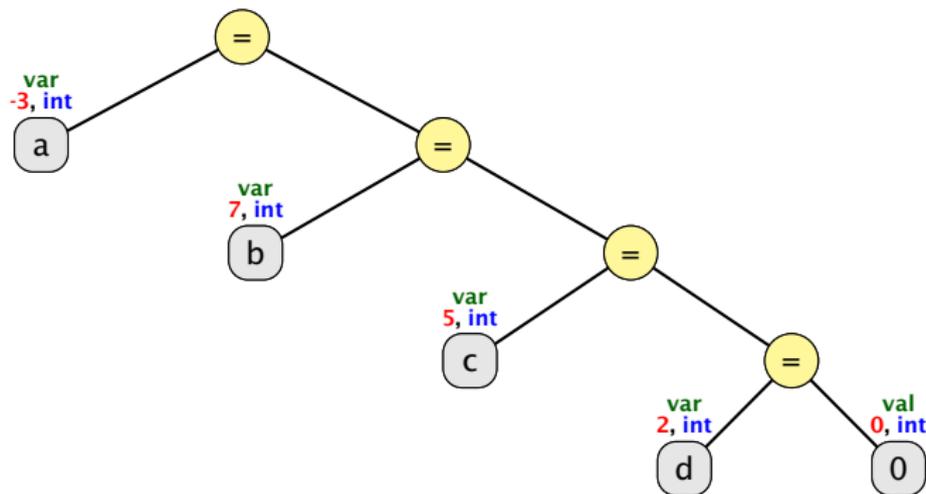
a [-3]

b [7]

c [5]

d [2]

Beispiel: $a = b = c = d = 0$



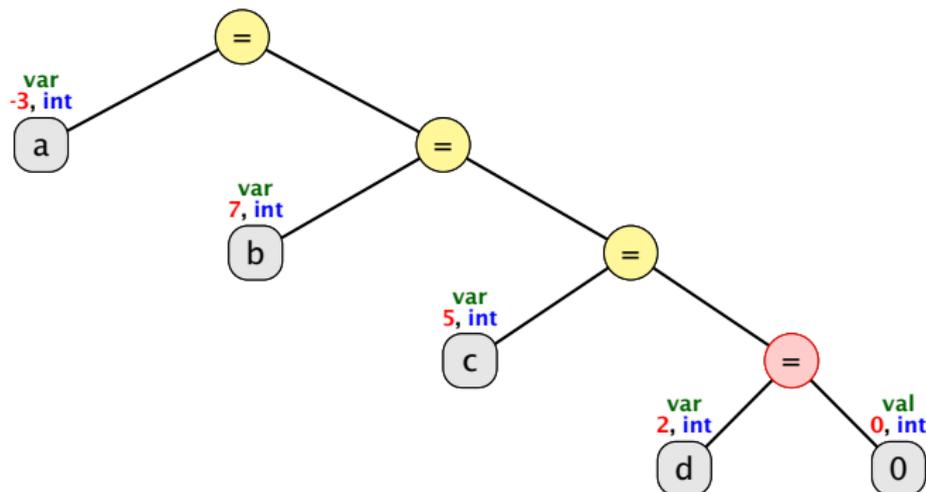
a [-3]

b [7]

c [5]

d [2]

Beispiel: $a = b = c = d = 0$



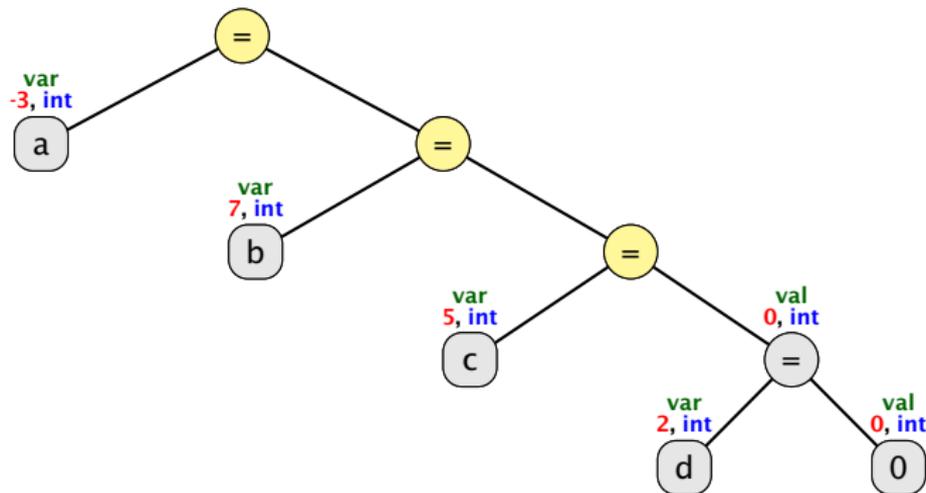
a

b

c

d

Beispiel: $a = b = c = d = 0$



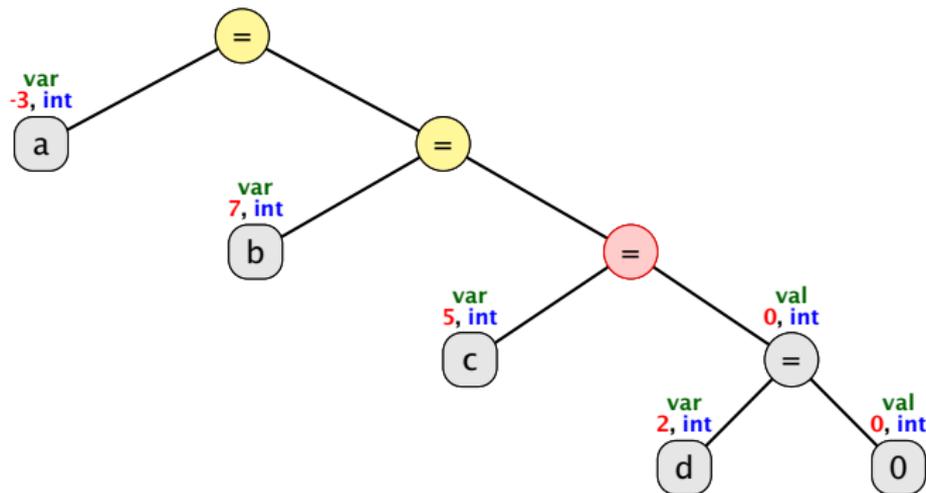
a

b

c

d

Beispiel: $a = b = c = d = 0$



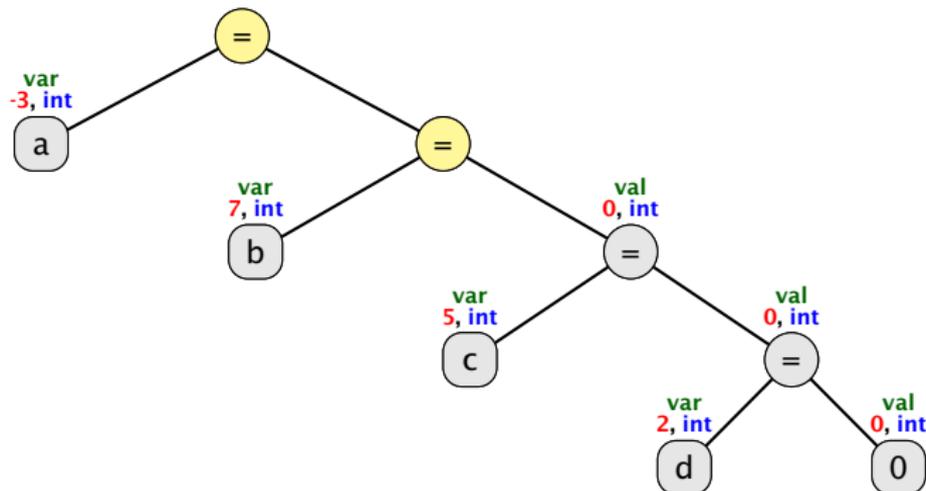
a

b

c

d

Beispiel: $a = b = c = d = 0$



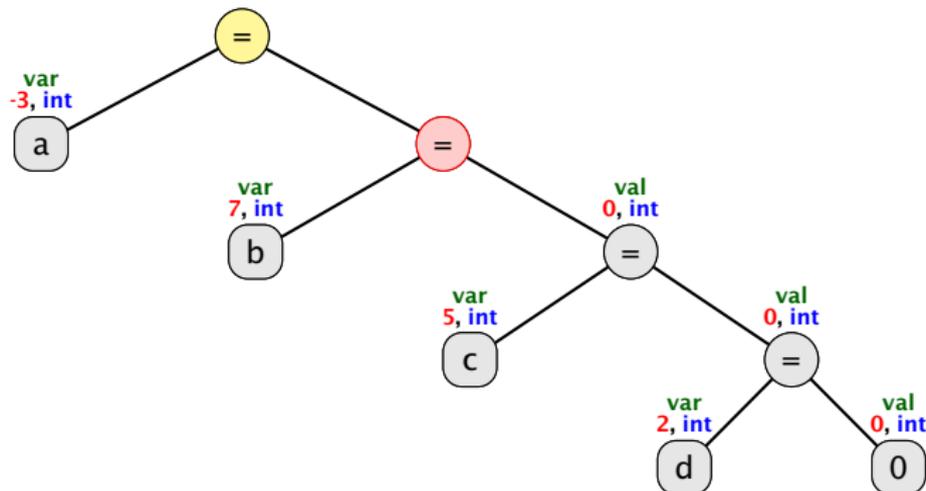
a [-3]

b [7]

c [0]

d [0]

Beispiel: $a = b = c = d = 0$



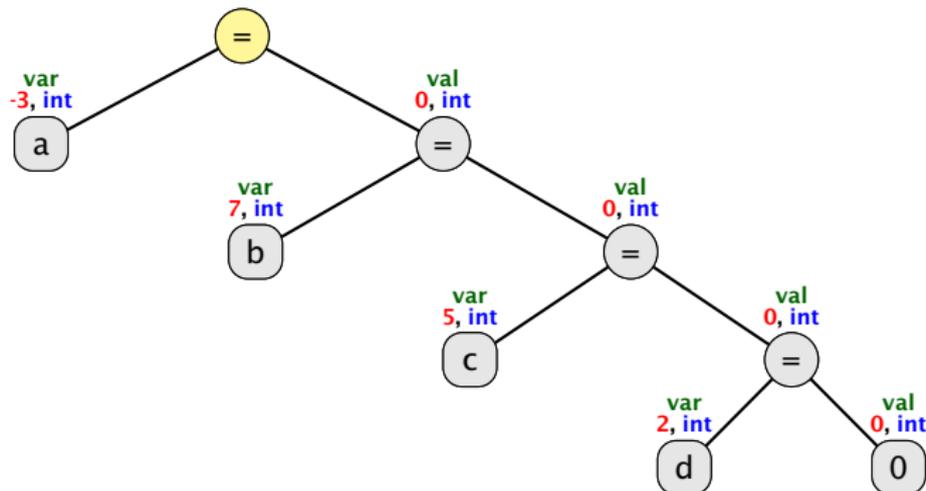
a

b

c

d

Beispiel: $a = b = c = d = 0$



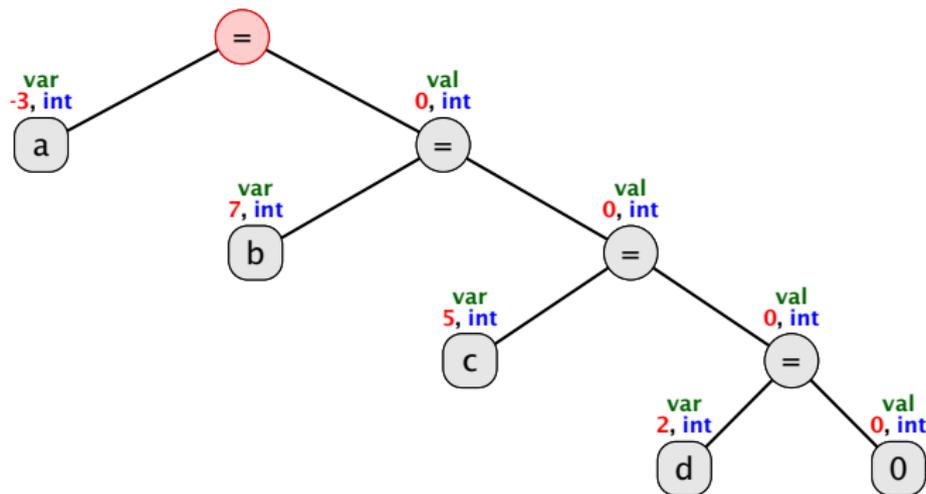
a [-3]

b [0]

c [0]

d [0]

Beispiel: $a = b = c = d = 0$



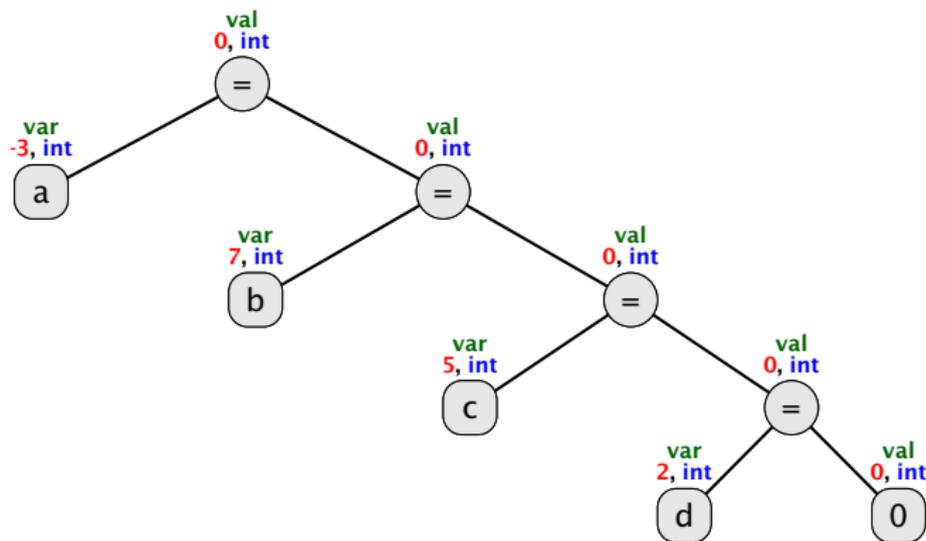
a

b

c

d

Beispiel: $a = b = c = d = 0$



a 0

b 0

c 0

d 0

Beispiel: $a \neq 0 \ \&\& \ b/a < 10$

a != 0 && b / a < 10

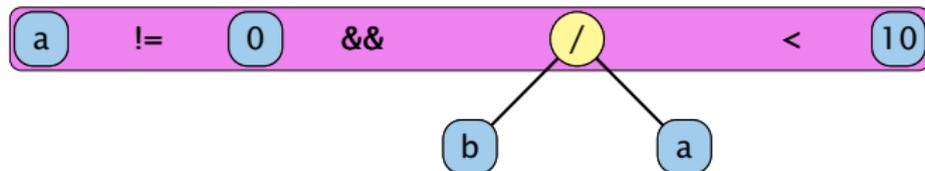
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



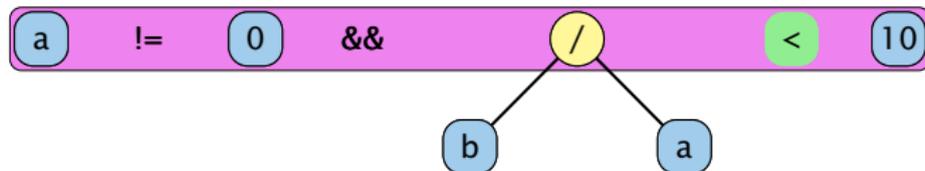
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



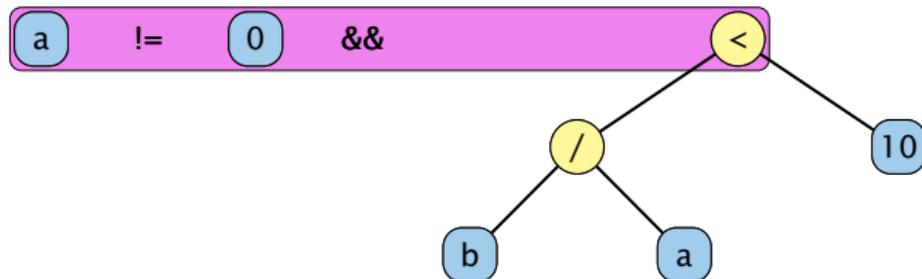
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



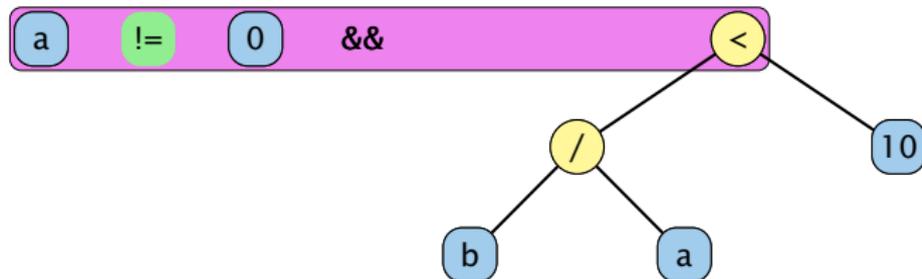
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



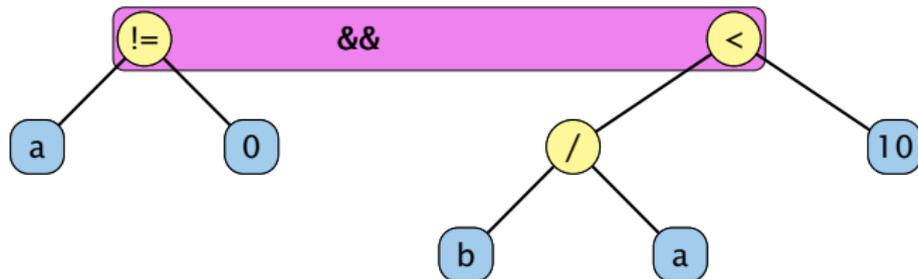
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



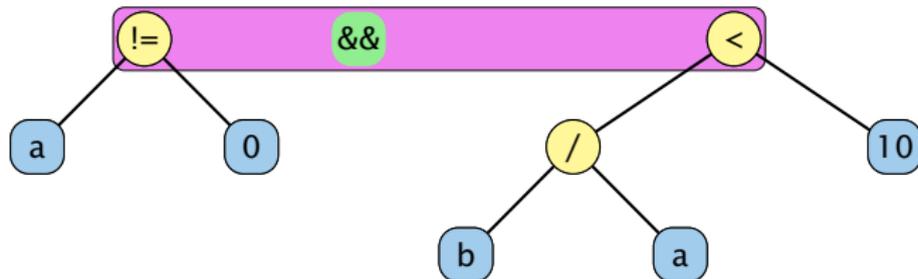
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



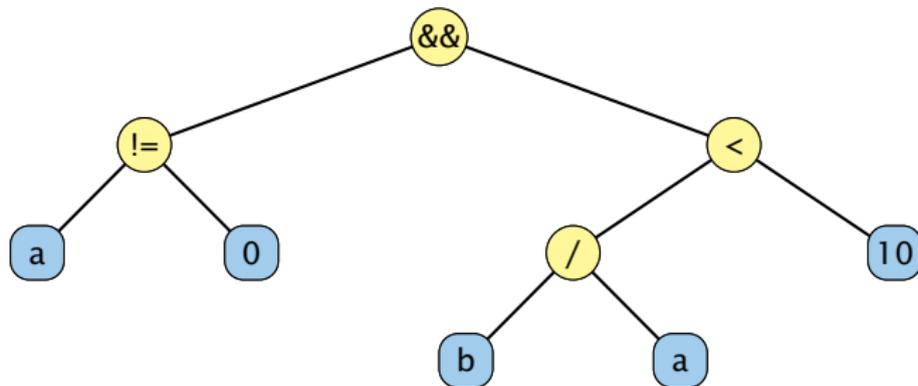
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



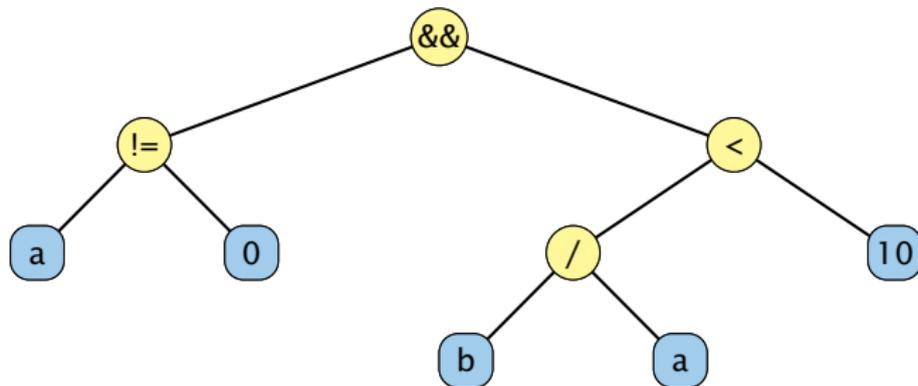
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



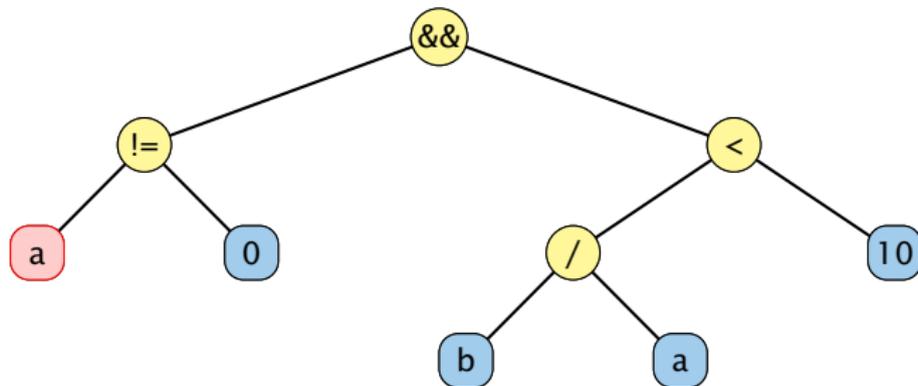
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

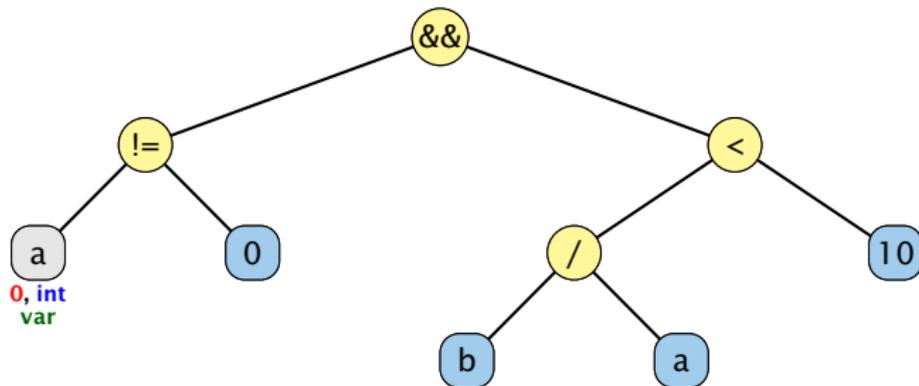
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a

b

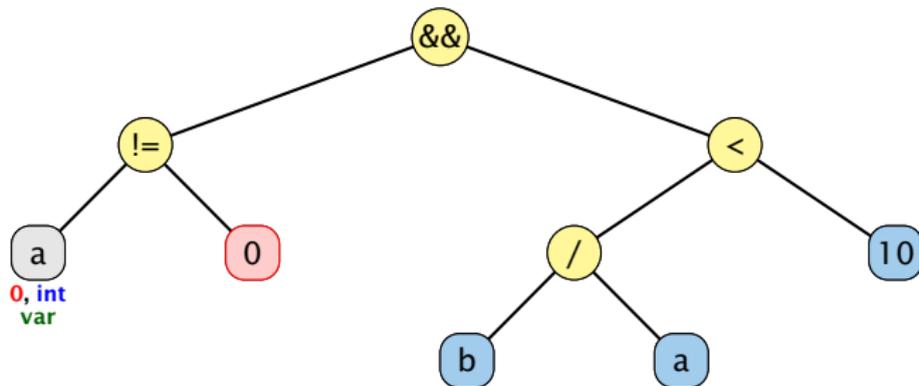
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

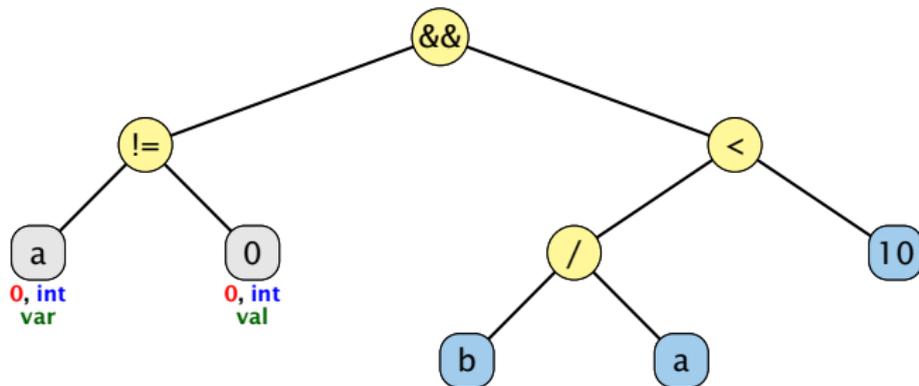
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

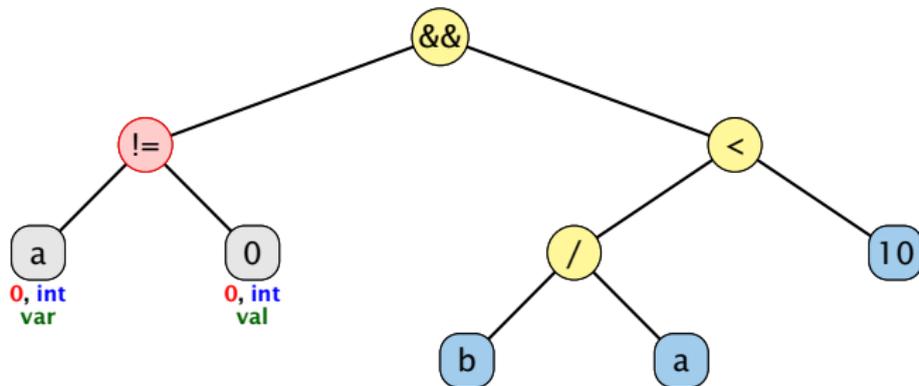
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

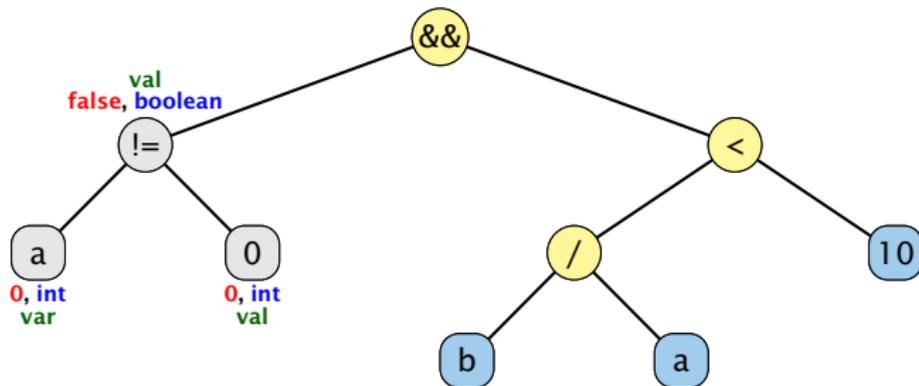
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a

b

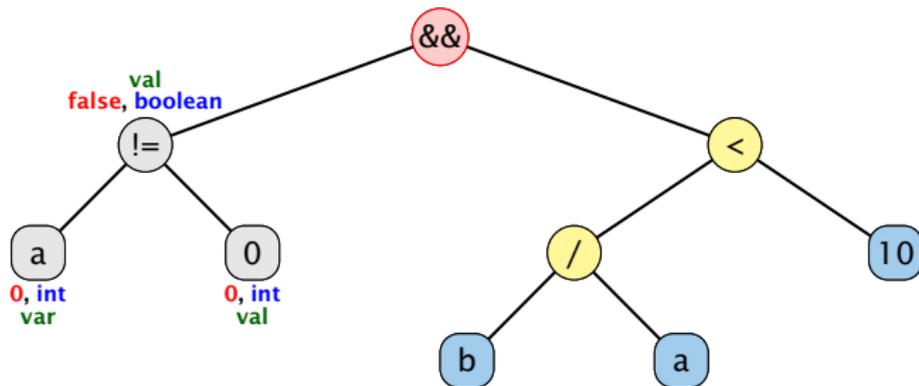
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

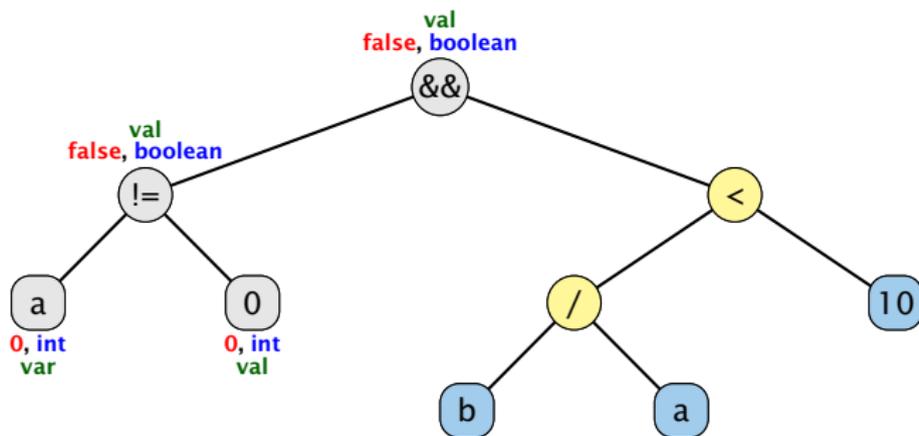
Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

Beispiel: $a \neq 0 \ \&\& \ b/a < 10$



a 0

b 4

Beispiel: $y = x + ++x$

y = x + ++x

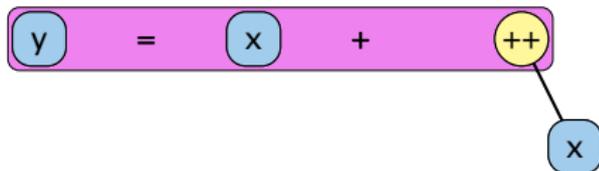
Beispiel: $y = x + ++x$



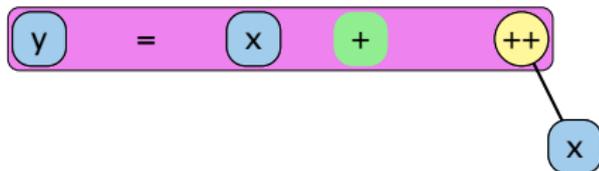
Beispiel: $y = x + ++x$



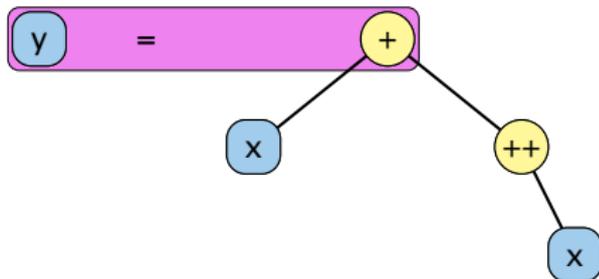
Beispiel: $y = x + ++x$



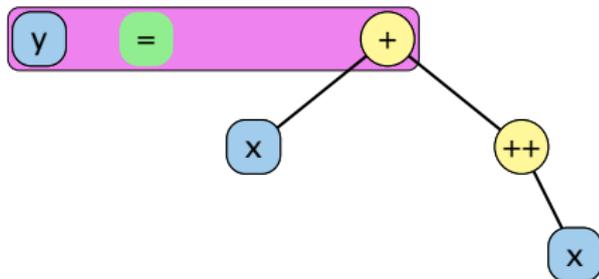
Beispiel: $y = x + ++x$



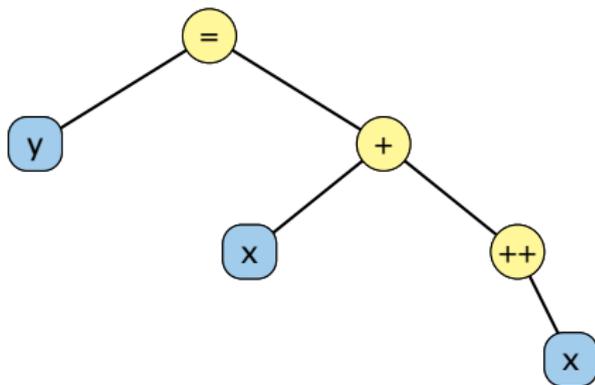
Beispiel: $y = x + ++x$



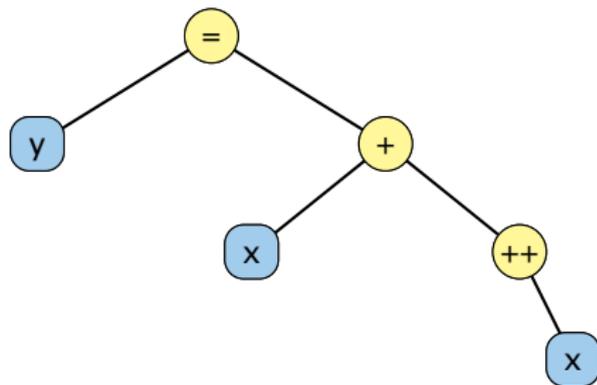
Beispiel: $y = x + ++x$



Beispiel: $y = x + ++x$



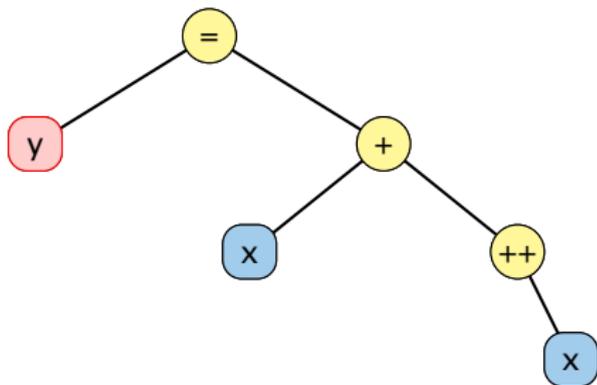
Beispiel: $y = x + ++x$



x 4

y 0

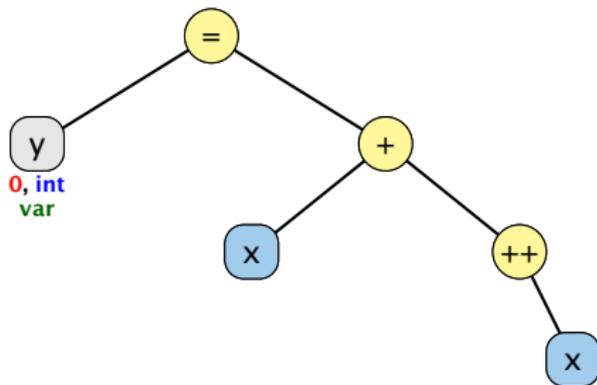
Beispiel: $y = x + ++x$



x 4

y 0

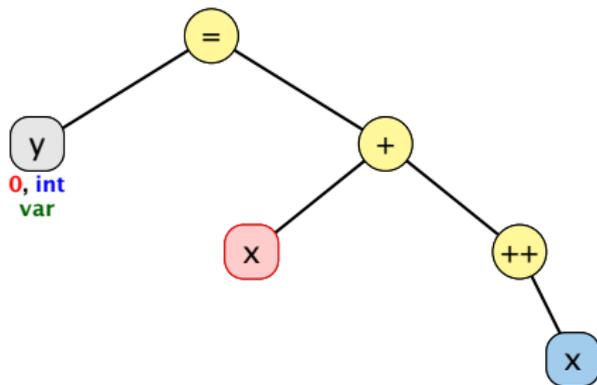
Beispiel: $y = x + ++x$



x 4

y 0

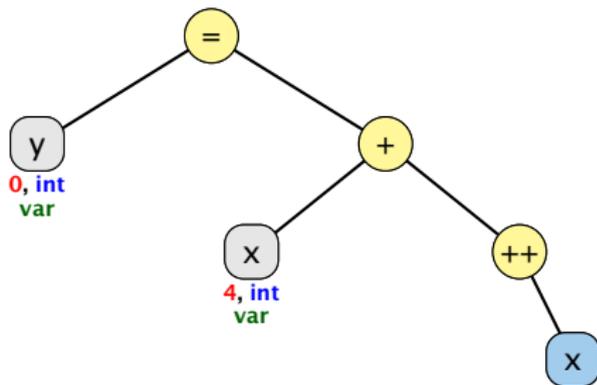
Beispiel: $y = x + ++x$



x 4

y 0

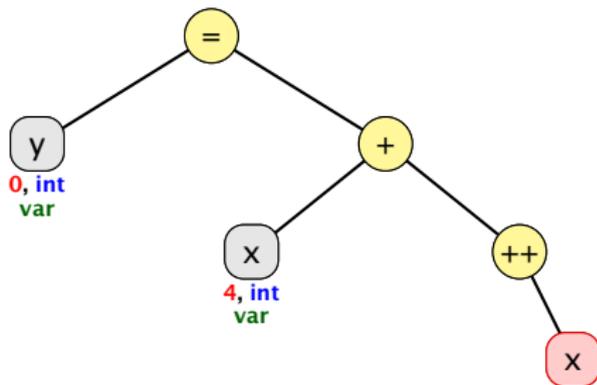
Beispiel: $y = x + ++x$



x 4

y 0

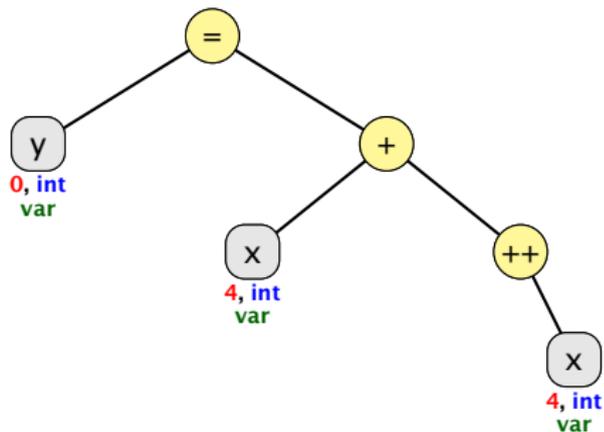
Beispiel: $y = x + ++x$



x 4

y 0

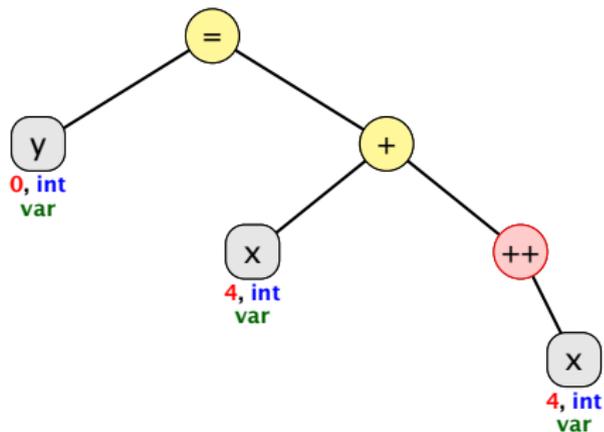
Beispiel: $y = x + ++x$



x 4

y 0

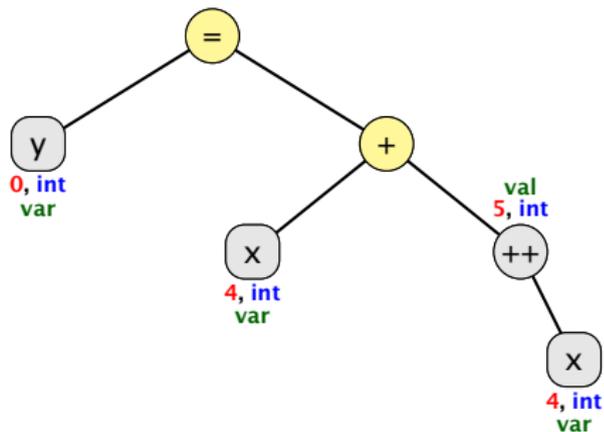
Beispiel: $y = x + ++x$



x 4

y 0

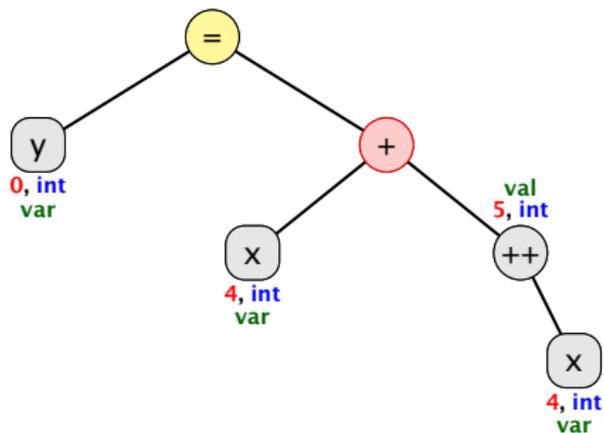
Beispiel: $y = x + ++x$



x 5

y 0

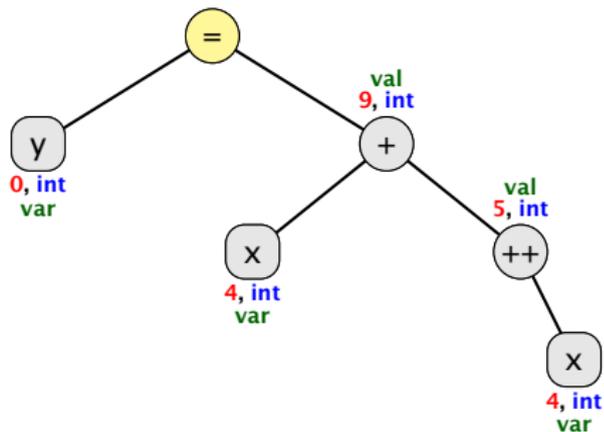
Beispiel: $y = x + ++x$



x 5

y 0

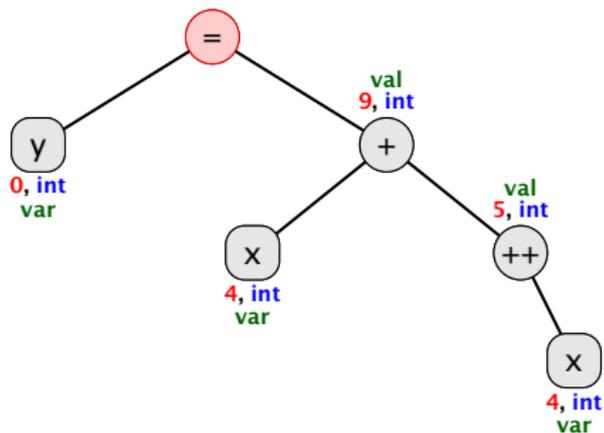
Beispiel: $y = x + ++x$



x 5

y 0

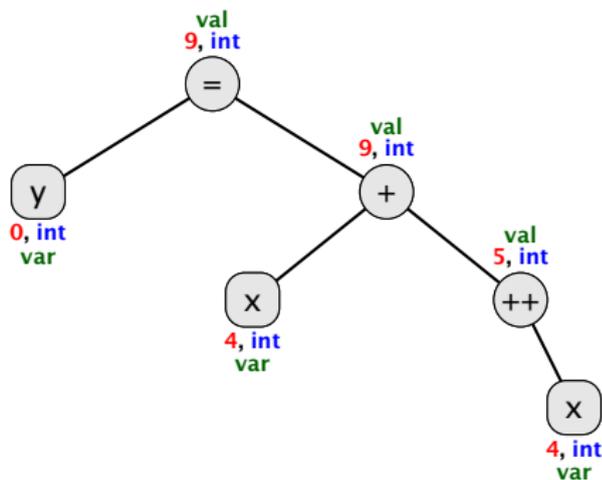
Beispiel: $y = x + ++x$



x 5

y 0

Beispiel: $y = x + ++x$



x 5

y 9

Beispiel: $y = x++ + x$

y = x ++ + x

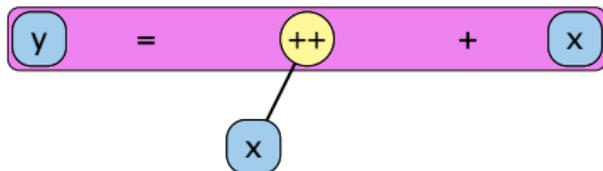
Beispiel: $y = x++ + x$



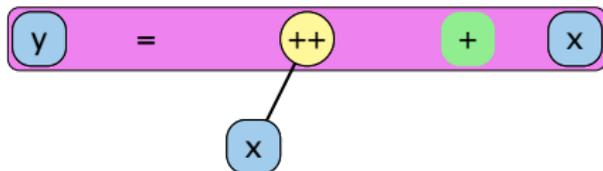
Beispiel: $y = x++ + x$



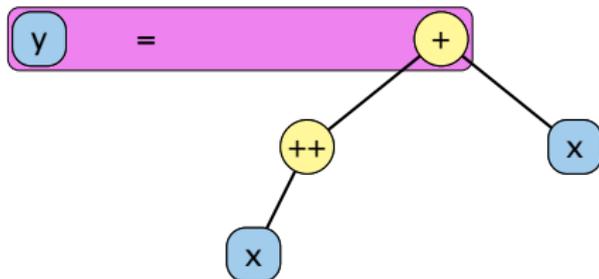
Beispiel: $y = x++ + x$



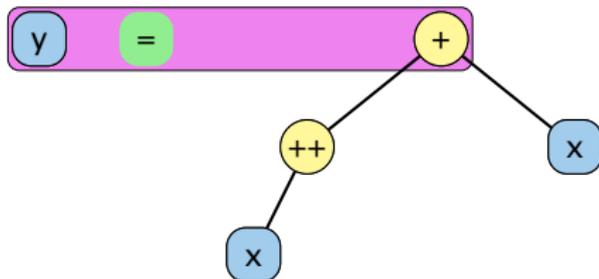
Beispiel: $y = x++ + x$



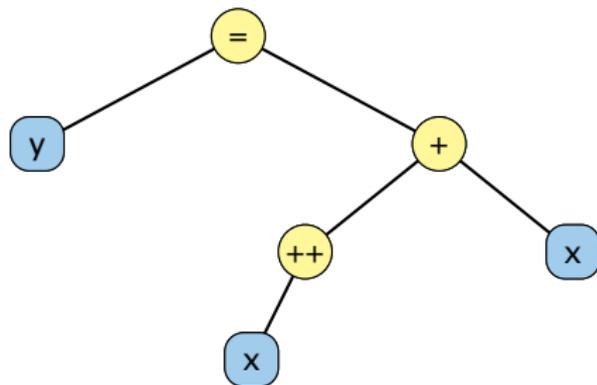
Beispiel: $y = x++ + x$



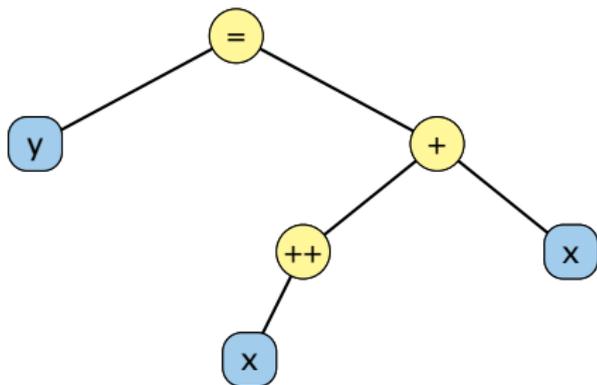
Beispiel: $y = x++ + x$



Beispiel: $y = x++ + x$



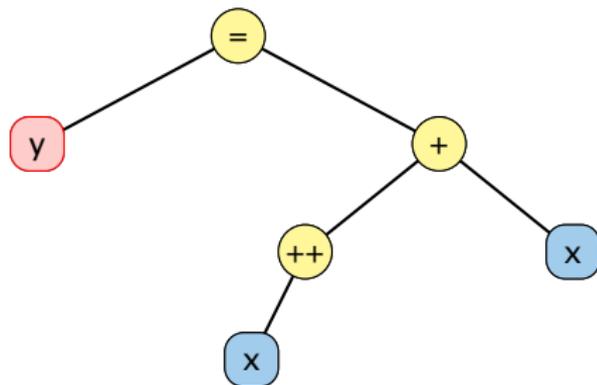
Beispiel: $y = x++ + x$



x 4

y 0

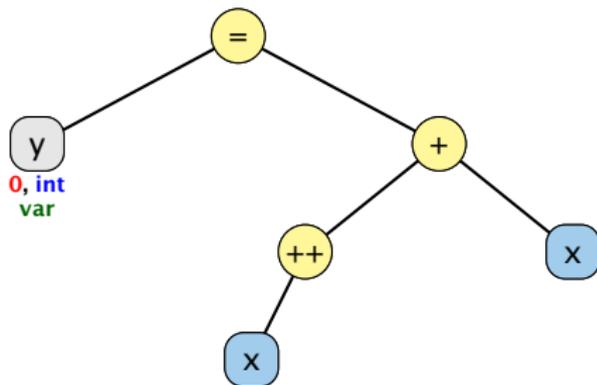
Beispiel: $y = x++ + x$



x 4

y 0

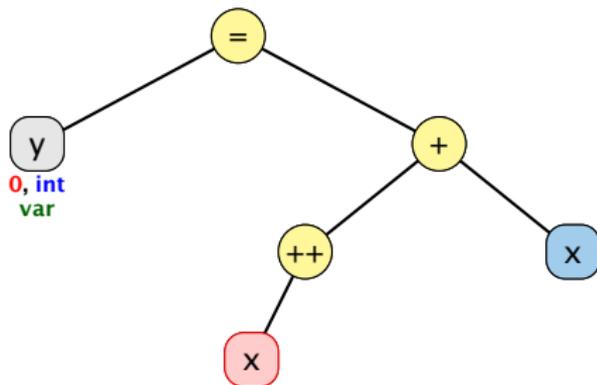
Beispiel: $y = x++ + x$



x

y

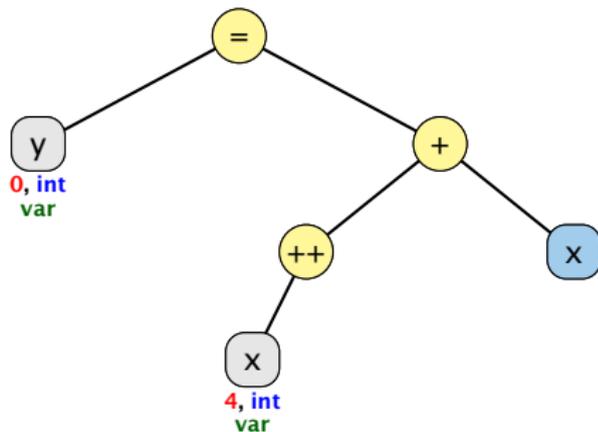
Beispiel: $y = x++ + x$



x 4

y 0

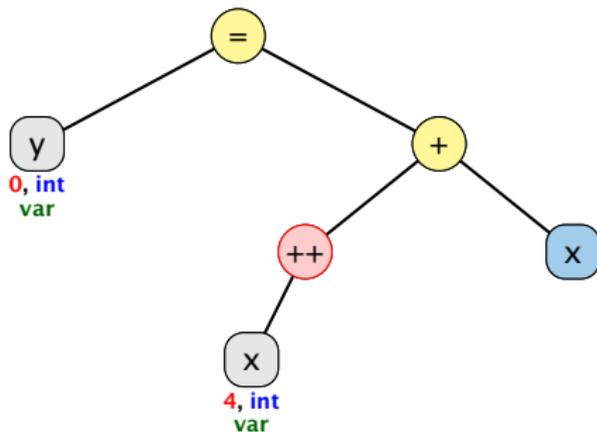
Beispiel: $y = x++ + x$



x 4

y 0

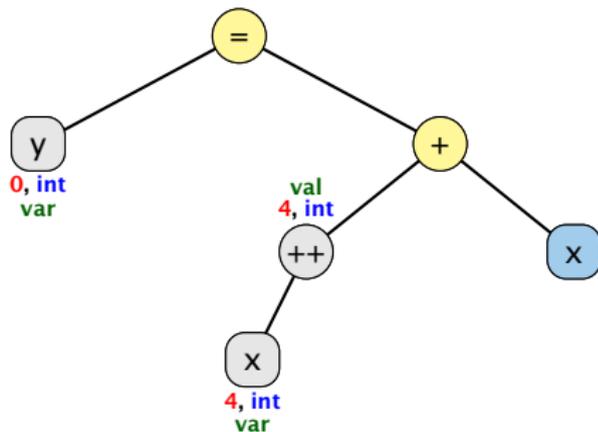
Beispiel: $y = x++ + x$



x 4

y 0

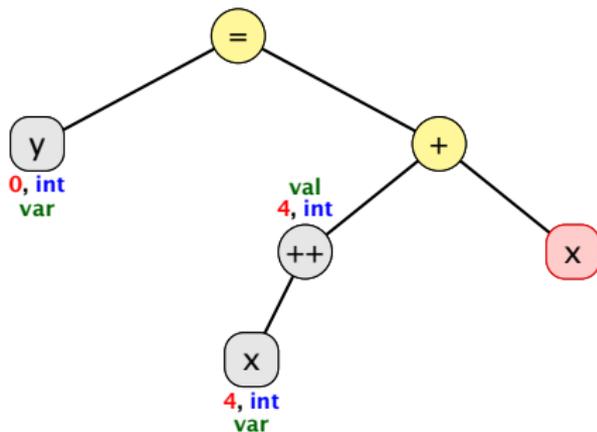
Beispiel: $y = x++ + x$



x 5

y 0

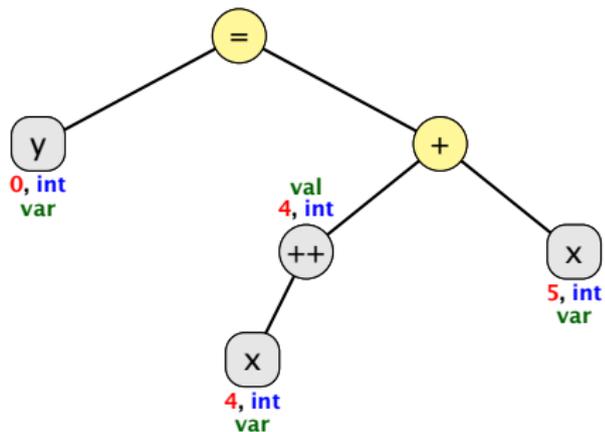
Beispiel: $y = x++ + x$



x 5

y 0

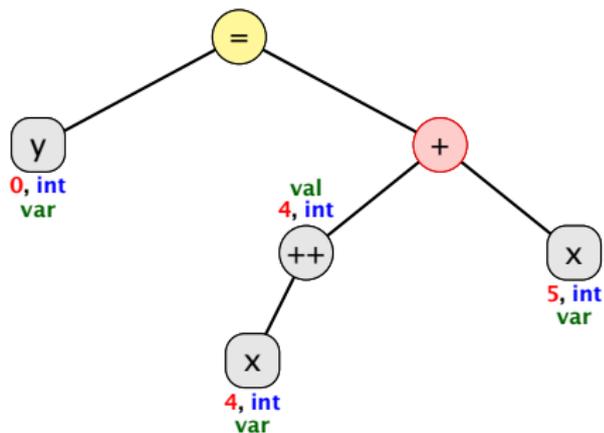
Beispiel: $y = x++ + x$



x 5

y 0

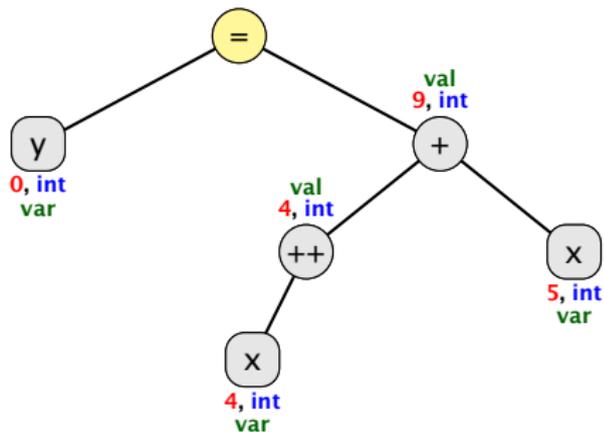
Beispiel: $y = x++ + x$



x 5

y 0

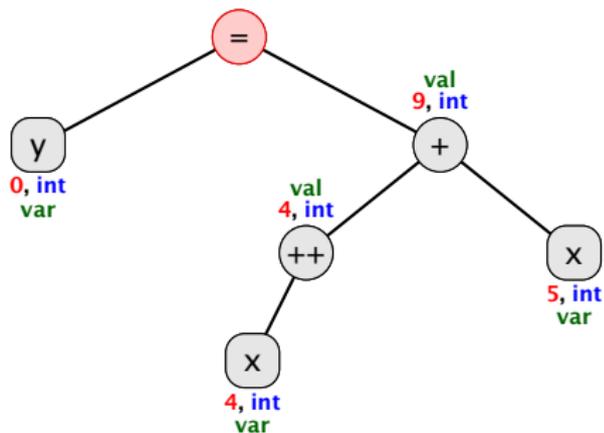
Beispiel: $y = x++ + x$



x 5

y 0

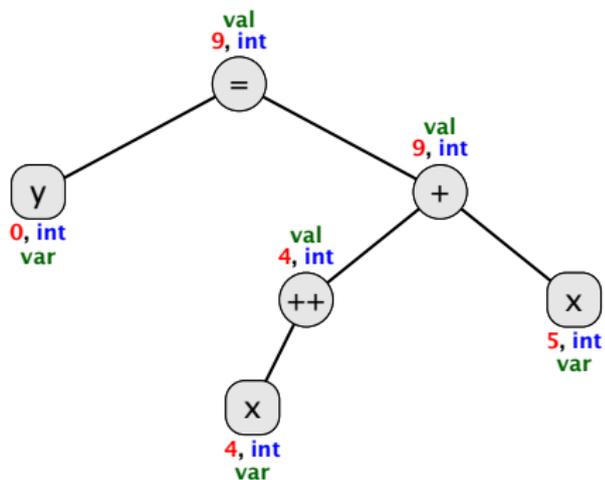
Beispiel: $y = x++ + x$



x 5

y 0

Beispiel: $y = x++ + x$



x 5

y 9

Impliziter Typecast

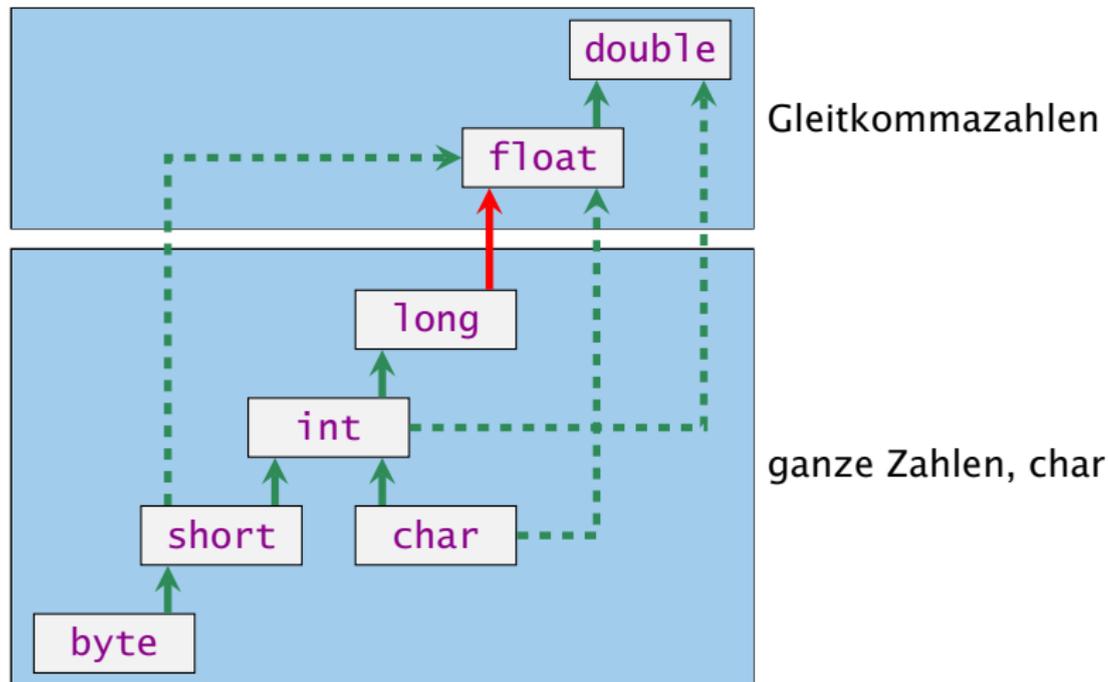
Wenn ein Ausdruck vom **TypA** an einer Stelle verwendet wird, wo ein Ausdruck vom **TypB** erforderlich ist, wird

- ▶ entweder der Ausdruck vom **TypA** in einen Ausdruck vom **TypB** **gecastet** (**impliziter Typecast**),
- ▶ oder ein Compilerfehler erzeugt, falls dieser Cast nicht (automatisch) erlaubt ist.

Beispiel: Zuweisung

```
long x = 5;  
int y = 3;  
x = y; // impliziter Cast von int nach long
```

Erlaubte Implizite Typecasts - Numerische Typen



Konvertierung von `long` nach `double` oder von `int` nach `float` kann Information verlieren wird aber **automatisch** durchgeführt.

Welcher Typ wird benötigt?

Operatoren sind üblicherweise **überladen**, d.h. ein Symbol (+, -, ...) steht in Abhängigkeit der Parameter (Argumente) für unterschiedliche Funktionen.

+ : int \rightarrow int

+ : long \rightarrow long

+ : float \rightarrow float

+ : double \rightarrow double

+ : int \times int \rightarrow int

+ : long \times long \rightarrow long

+ : float \times float \rightarrow float

+ : double \times double \rightarrow double

+ : String \times String \rightarrow String

Der Compiler muss in der Lage sein **während der Compilierung** die richtige Funktion zu bestimmen.

Impliziter Typecast

Der Compiler wertet nur die Typen des Ausdrucksbaums aus.

- ▶ Für jeden inneren Knoten wählt er dann die geeignete Funktion (z.B. $+ : \text{long} \times \text{long} \rightarrow \text{long}$ falls ein $+$ -Knoten zwei long -Argumente erhält).
- ▶ Falls keine passende Funktion gefunden wird, versucht der Compiler durch **implizite Typecasts** die Operanden an eine Funktion anzupassen.
- ▶ Dies geschieht auch für selbstgeschriebene Funktionen (z.B. $\text{min}(\text{int } a, \text{int } b)$ und $\text{min}(\text{long } a, \text{long } b)$).
- ▶ Der Compiler nimmt die Funktion mit der speziellsten **Signatur**.

Speziellste Signatur



Ordnungsrelationen

Relation \preceq : $\text{TypA} \preceq \text{TypB}$ falls TypA nach TypB (implizit) gecasted werden kann:

- ▶ **reflexiv:** $T \preceq T$
- ▶ **transitiv:** $T_1 \preceq T_2 \wedge T_2 \preceq T_3 \Rightarrow T_1 \preceq T_3$
- ▶ **antisymmetrisch:** $T_1 \preceq T_2 \wedge T_2 \preceq T_1 \Rightarrow T_1 = T_2$

d.h., \preceq definiert **Halbordnung auf der Menge der Typen**.

Relation \preceq_k : $(T_1, \dots, T_k) \preceq_k (T'_1, \dots, T'_k)$ falls $T_i \preceq T'_i$ für alle $i \in \{1, \dots, k\}$:

- ▶ **reflexiv:** $\mathcal{T} \preceq_k \mathcal{T}$
- ▶ **transitiv:** $\mathcal{T}_1 \preceq_k \mathcal{T}_2 \wedge \mathcal{T}_2 \preceq_k \mathcal{T}_3 \Rightarrow \mathcal{T}_1 \preceq_k \mathcal{T}_3$
- ▶ **antisymmetrisch:** $\mathcal{T}_1 \preceq_k \mathcal{T}_2 \wedge \mathcal{T}_2 \preceq_k \mathcal{T}_1 \Rightarrow \mathcal{T}_1 = \mathcal{T}_2$

d.h., \preceq_k definiert **Halbordnung auf Menge der k -Tupel von Typen**

Ordnungsrelationen

Relation \preceq : $\text{TypA} \preceq \text{TypB}$ falls TypA nach TypB (implizit) gecasted werden kann:

- ▶ **reflexiv:** $T \preceq T$
- ▶ **transitiv:** $T_1 \preceq T_2 \wedge T_2 \preceq T_3 \Rightarrow T_1 \preceq T_3$
- ▶ **antisymmetrisch:** $T_1 \preceq T_2 \wedge T_2 \preceq T_1 \Rightarrow T_1 = T_2$

d.h., \preceq definiert **Halbordnung auf der Menge der Typen**.

Relation \preceq_k : $(T_1, \dots, T_k) \preceq_k (T'_1, \dots, T'_k)$ falls $T_i \preceq T'_i$ für alle $i \in \{1, \dots, k\}$:

- ▶ **reflexiv:** $\mathcal{T} \preceq_k \mathcal{T}$
- ▶ **transitiv:** $\mathcal{T}_1 \preceq_k \mathcal{T}_2 \wedge \mathcal{T}_2 \preceq_k \mathcal{T}_3 \Rightarrow \mathcal{T}_1 \preceq_k \mathcal{T}_3$
- ▶ **antisymmetrisch:** $\mathcal{T}_1 \preceq_k \mathcal{T}_2 \wedge \mathcal{T}_2 \preceq_k \mathcal{T}_1 \Rightarrow \mathcal{T}_1 = \mathcal{T}_2$

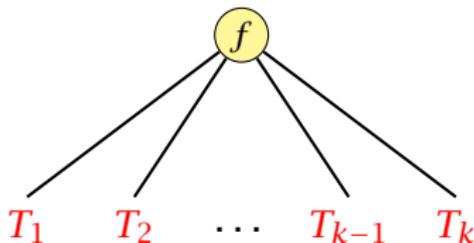
d.h., \preceq_k definiert **Halbordnung auf Menge der k -Tupel von Typen**

$R_1 \quad f(\mathcal{T}_1)$

$R_2 \quad f(\mathcal{T}_2)$

\vdots

$R_\ell \quad f(\mathcal{T}_\ell)$



$\mathcal{T}_1, \dots, \mathcal{T}_\ell$ sind k -Tupel von Typen für die eine Definition von f existiert.

$\mathcal{T} = (\mathcal{T}_1, \dots, \mathcal{T}_k)$ ist das k -tupel von Typen mit dem f aufgerufen wird.

Menge aller möglichen Funktionen/Tupel:

$$M := \{\mathcal{T}_i \mid \mathcal{T} \preceq_k \mathcal{T}_i\} .$$

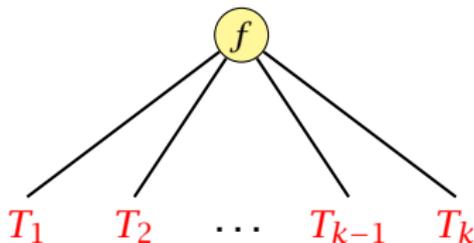
Wähle **kleinstes** Element aus M falls M ein eindeutig kleinstes Element besitzt (sonst Compilerfehler).

$R_1 \quad f(\mathcal{T}_1)$

$R_2 \quad f(\mathcal{T}_2)$

\vdots

$R_\ell \quad f(\mathcal{T}_\ell)$



$\mathcal{T}_1, \dots, \mathcal{T}_\ell$ sind k -Tupel von Typen für die eine Definition von f existiert.

$\mathcal{T} = (T_1, \dots, T_k)$ ist das k -tupel von Typen mit dem f aufgerufen wird.

Menge aller möglichen Funktionen/Tupel:

$$M := \{\mathcal{T}_i \mid \mathcal{T} \preceq_k \mathcal{T}_i\} .$$

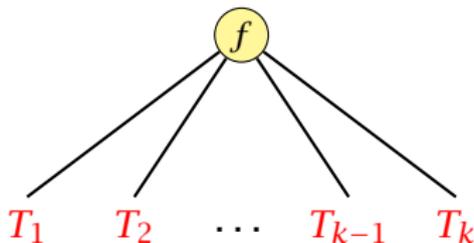
Wähle **kleinstes** Element aus M falls M ein eindeutig kleinstes Element besitzt (sonst Compilerfehler).

$R_1 \quad f(\mathcal{T}_1)$

$R_2 \quad f(\mathcal{T}_2)$

\vdots

$R_\ell \quad f(\mathcal{T}_\ell)$



$\mathcal{T}_1, \dots, \mathcal{T}_\ell$ sind k -Tupel von Typen für die eine Definition von f existiert.

$\mathcal{T} = (T_1, \dots, T_k)$ ist das k -tupel von Typen mit dem f aufgerufen wird.

Menge aller möglichen Funktionen/Tupel:

$$M := \{\mathcal{T}_i \mid \mathcal{T} \preceq_k \mathcal{T}_i\} .$$

Wähle **kleinstes** Element aus M falls M ein eindeutig kleinstes Element besitzt (sonst Compilerfehler).

Impliziter Typecast – Numerische Typen

Angenommen wir haben Funktionen

```
int min(int a, int b)
```

```
float min(float a, float b)
```

```
double min(double a, double b)
```

definiert.

```
1 long a = 7, b = 3;  
2 double d = min(a, b);
```

würde die Funktion `float min(float a, float b)` aufrufen.

Impliziter Typecast

Bei Ausdrücken mit Seiteneffekten (Zuweisungen, ++ , --) gelten andere Regeln:

Beispiel: Zuweisungen

= : $\text{byte}^* \times \text{byte} \rightarrow \text{byte}$
= : $\text{char}^* \times \text{char} \rightarrow \text{char}$
= : $\text{short}^* \times \text{short} \rightarrow \text{short}$
= : $\text{int}^* \times \text{int} \rightarrow \text{int}$
= : $\text{long}^* \times \text{long} \rightarrow \text{long}$
= : $\text{float}^* \times \text{float} \rightarrow \text{float}$
= : $\text{double}^* \times \text{double} \rightarrow \text{double}$

Es wird nur der Parameter konvertiert, der nicht dem Seiteneffekt unterliegt.

5.3 Auswertung von Ausdrücken

Der Funktionsaufrufoperator:

<i>symbol</i>	<i>name</i>	<i>types</i>	<i>L/R</i>	<i>level</i>
()	Funktionsaufruf	Funktionsname, *	links	1

Beispiel: $x = \min(a, \min(a,b) + 4L)$

$$x = \min (a , \min (a , b) + 4L)$$

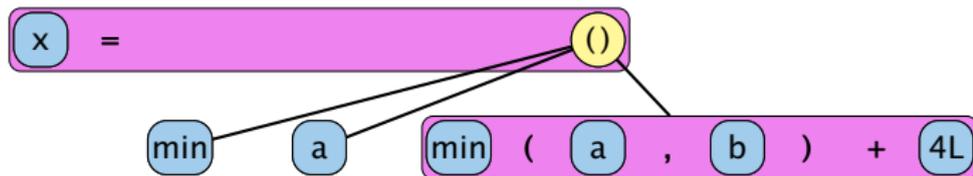
Beispiel: $x = \min(a, \min(a, b) + 4L)$

$$x = \min(a, \min(a, b) + 4L)$$

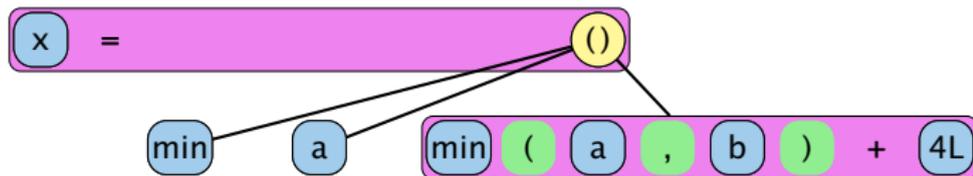
Beispiel: $x = \min(a, \min(a,b) + 4L)$

$$x = \min(a, \min(a, b) + 4L)$$

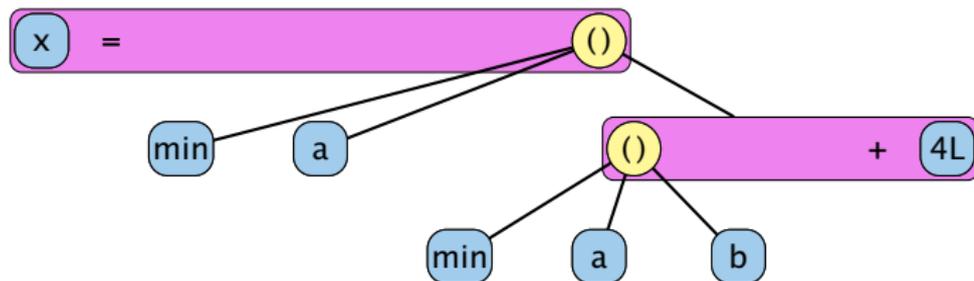
Beispiel: $x = \min(a, \min(a,b) + 4L)$



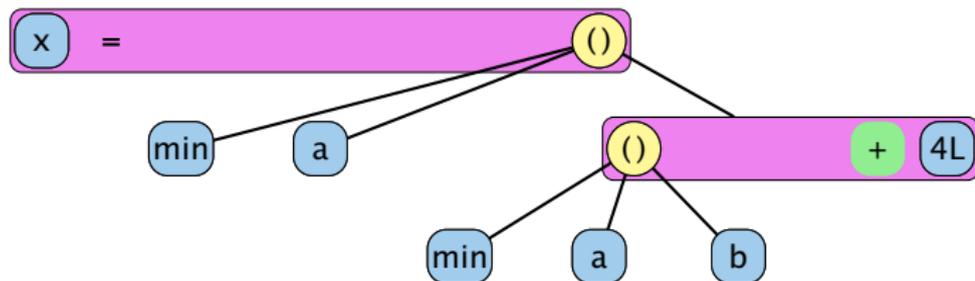
Beispiel: $x = \min(a, \min(a, b) + 4L)$



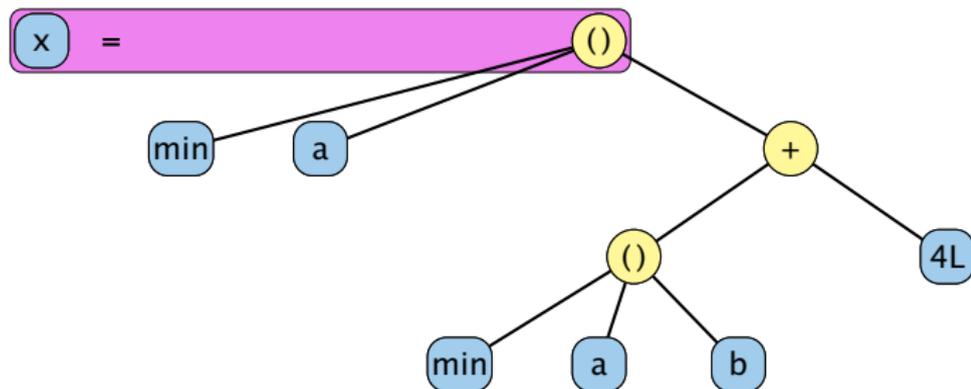
Beispiel: $x = \min(a, \min(a,b) + 4L)$



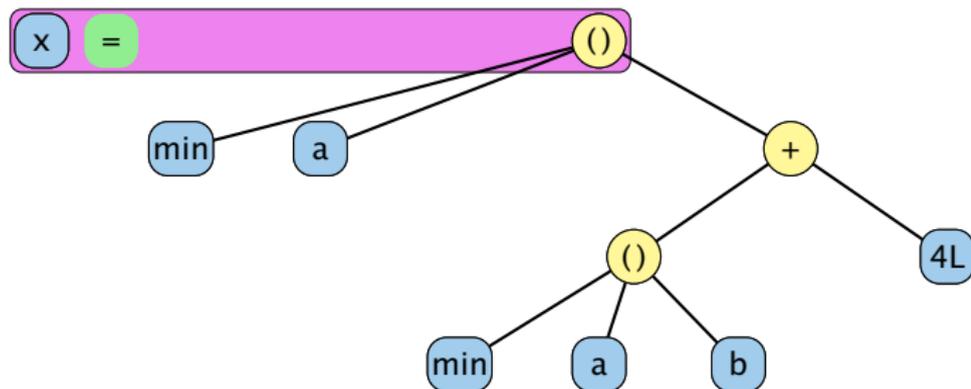
Beispiel: $x = \min(a, \min(a,b) + 4L)$



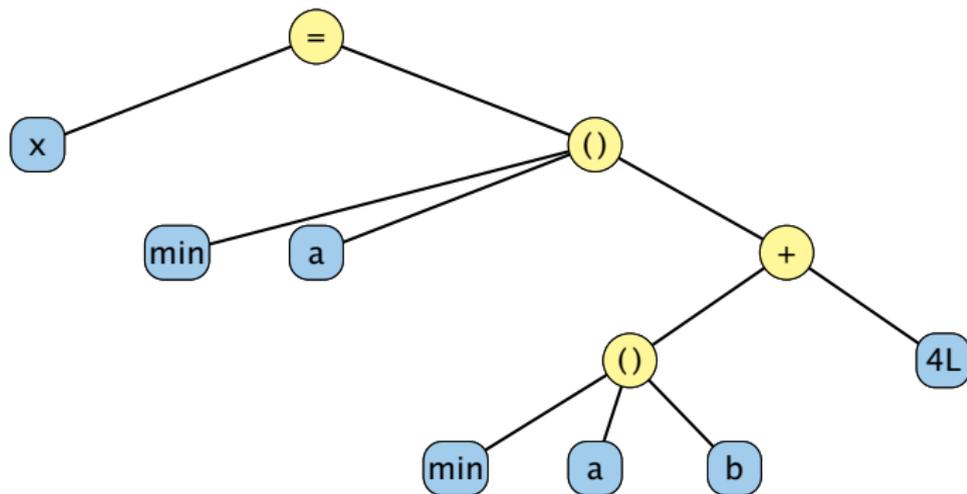
Beispiel: $x = \min(a, \min(a,b) + 4L)$



Beispiel: $x = \min(a, \min(a,b) + 4L)$



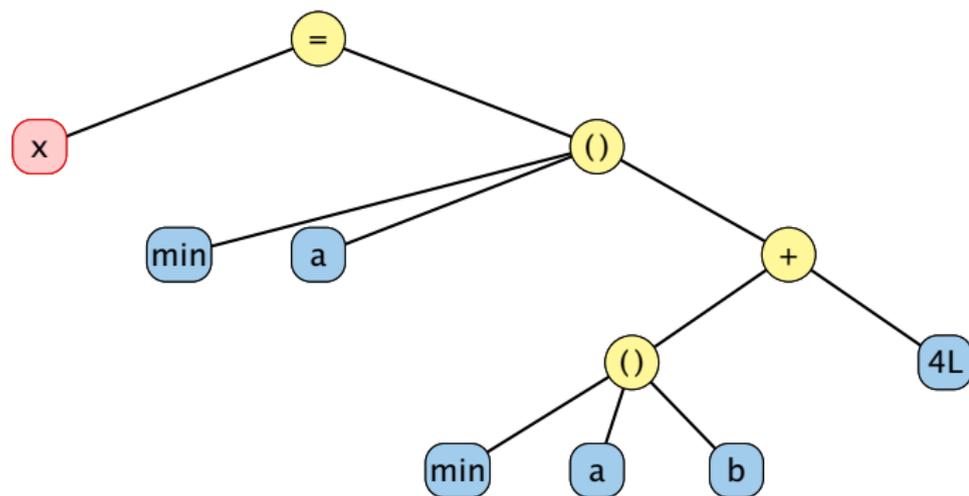
Beispiel: $x = \min(a, \min(a,b) + 4L)$



```
int min(int,int)
float min(float,float)
double min(double,double)
```

long x int a int b

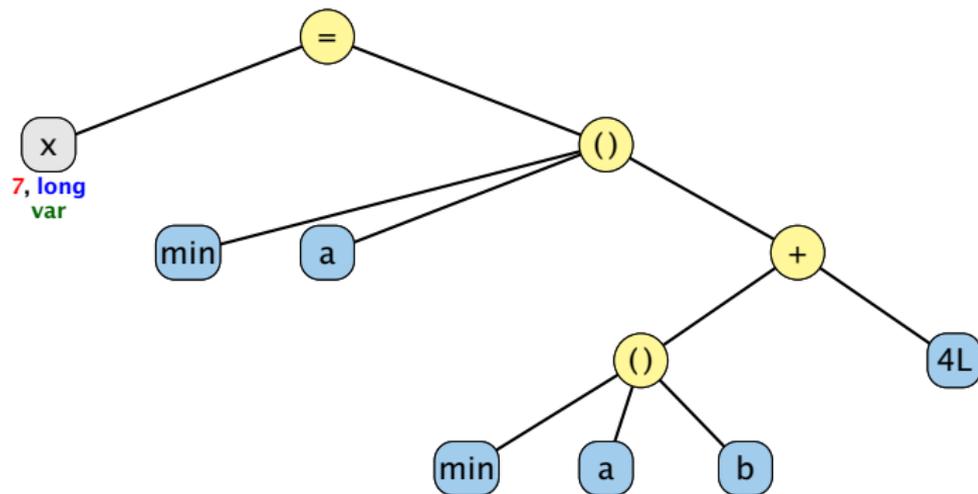
Beispiel: $x = \min(a, \min(a,b) + 4L)$



```
int min(int,int)
float min(float,float)
double min(double,double)
```

long x int a int b

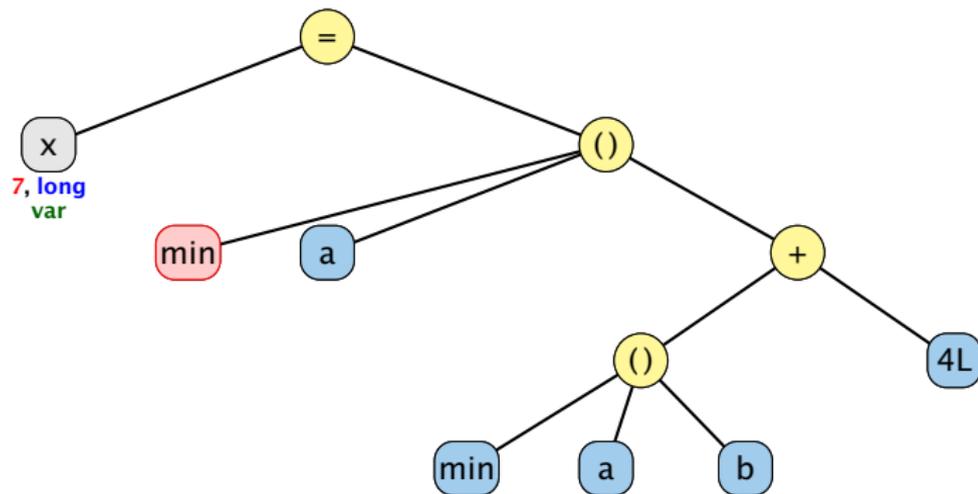
Beispiel: $x = \min(a, \min(a,b) + 4L)$



```
int min(int,int)
float min(float,float)
double min(double,double)
```

long x int a int b

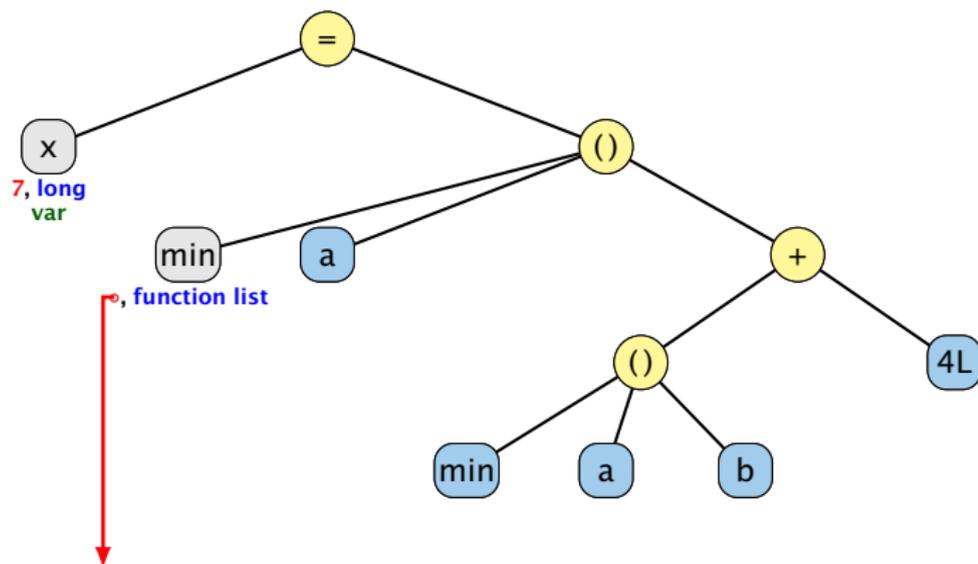
Beispiel: $x = \min(a, \min(a,b) + 4L)$



```
int min(int,int)
float min(float,float)
double min(double,double)
```

long x int a int b

Beispiel: $x = \min(a, \min(a,b) + 4L)$



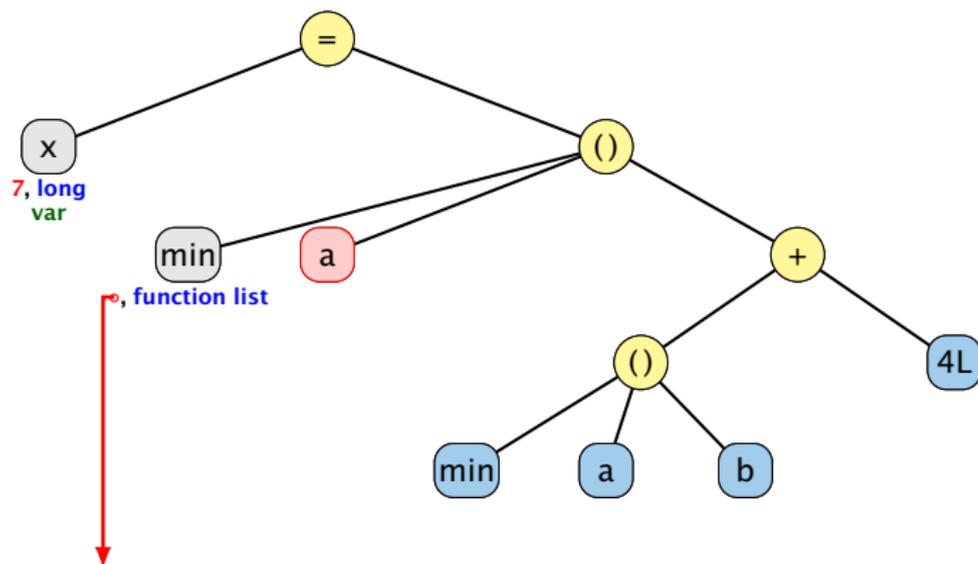
7, long
var

function list

int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

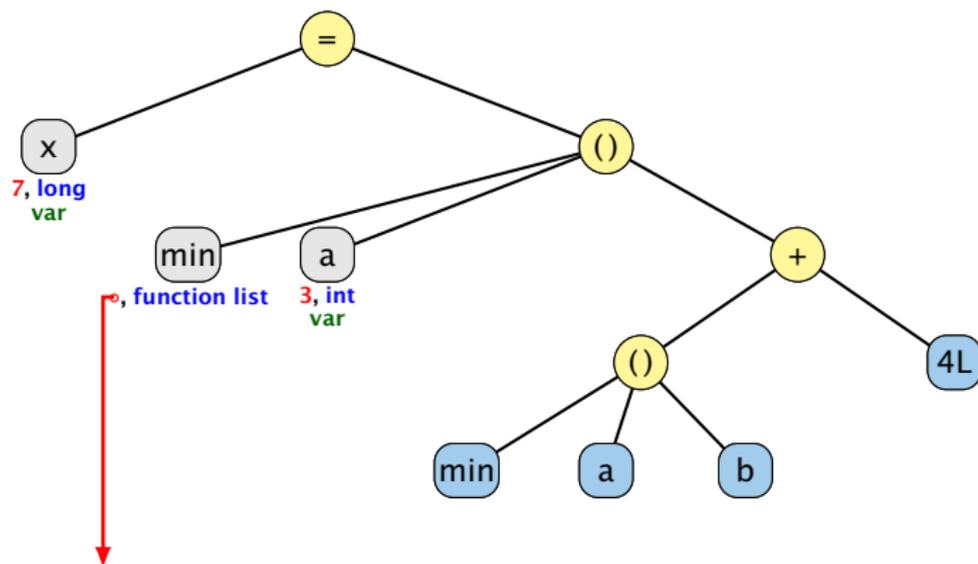
Beispiel: $x = \min(a, \min(a,b) + 4L)$



`int min(int,int)`
`float min(float,float)`
`double min(double,double)`

`long x` `int a` `int b`

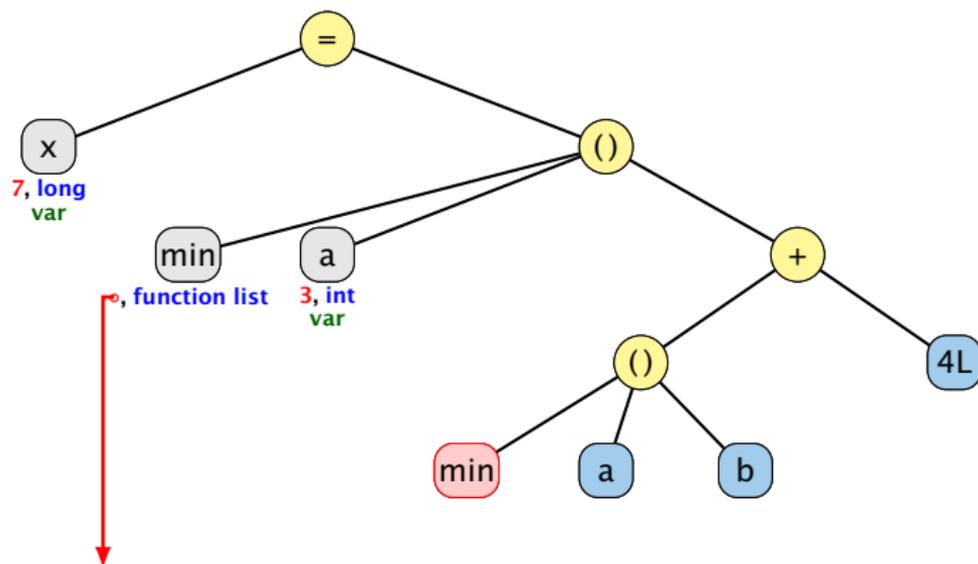
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

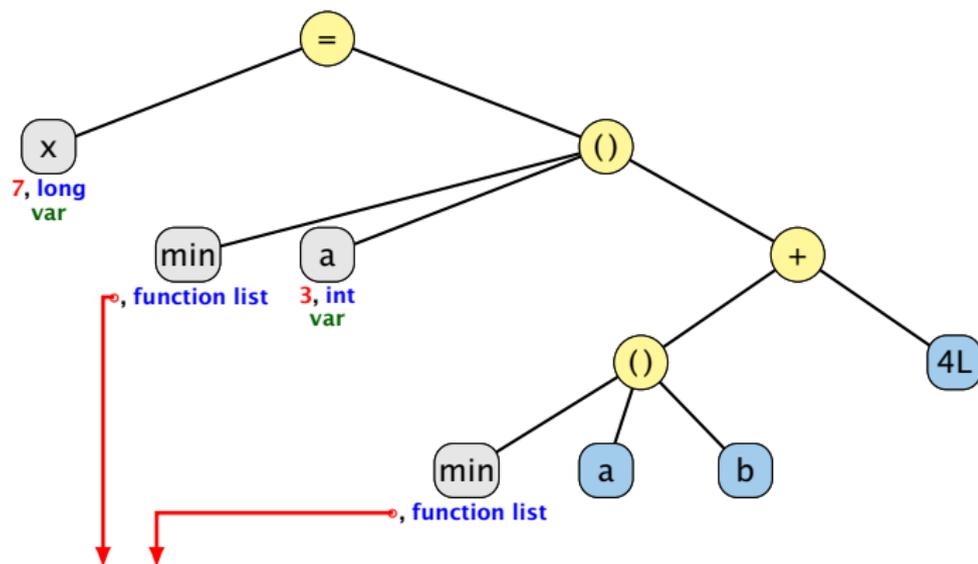
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

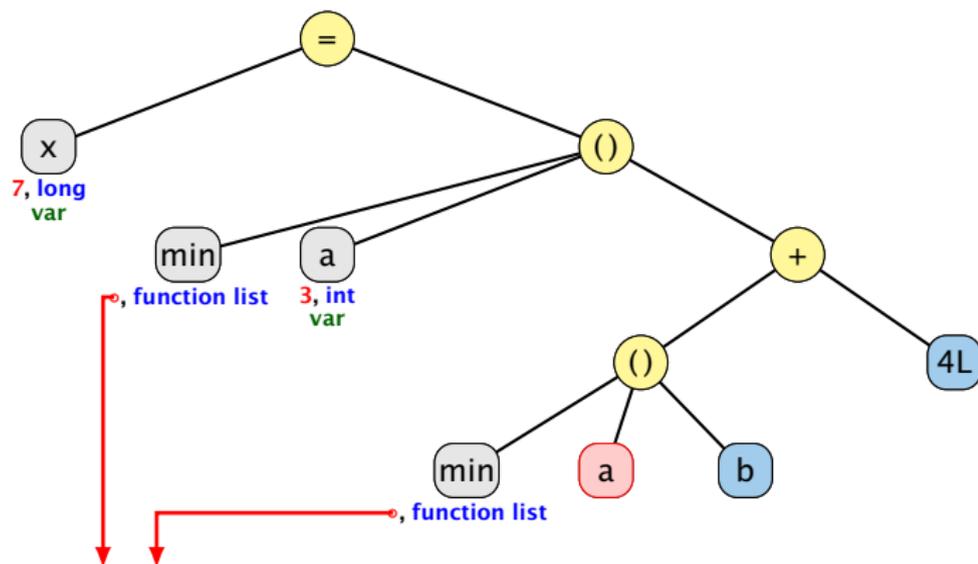
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x [7] int a [3] int b [5]

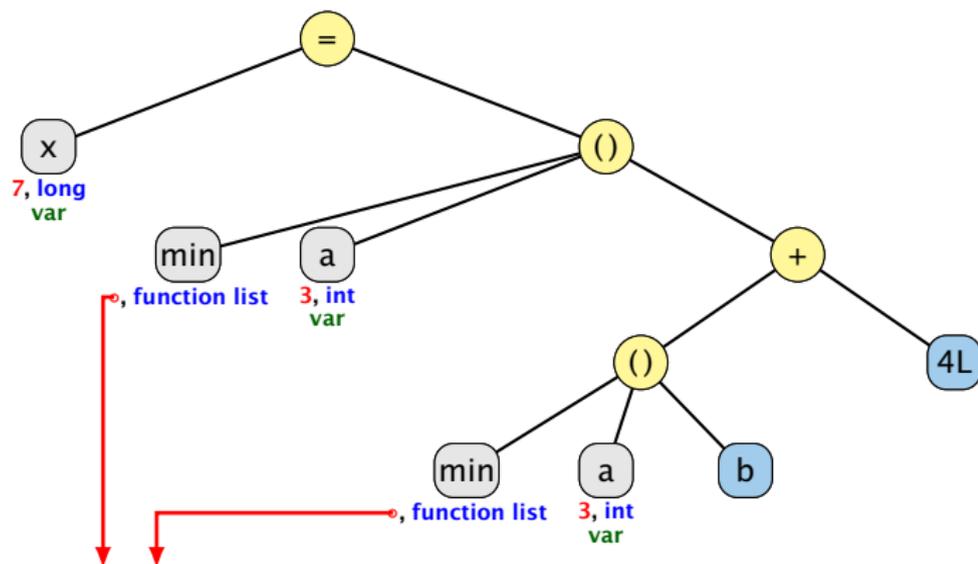
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x [7] int a [3] int b [5]

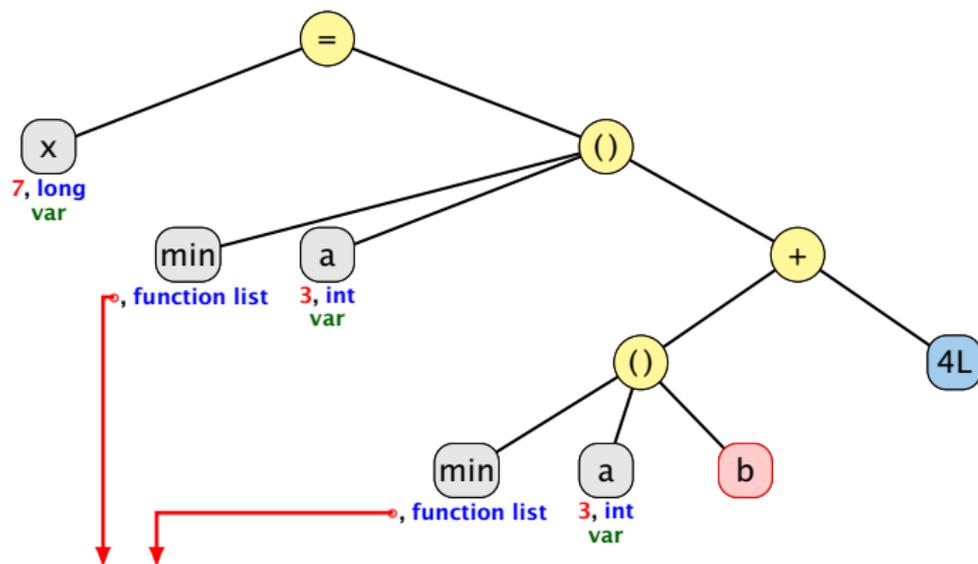
Beispiel: $x = \min(a, \min(a,b) + 4L)$



`int min(int,int)`
`float min(float,float)`
`double min(double,double)`

`long x` `int a` `int b`

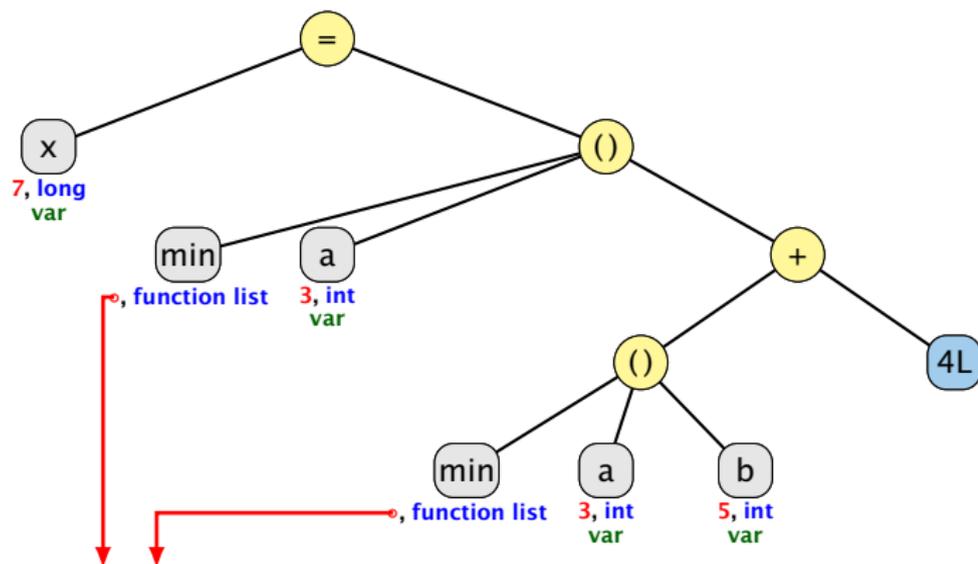
Beispiel: $x = \min(a, \min(a,b) + 4L)$



`int min(int,int)`
`float min(float,float)`
`double min(double,double)`

`long x` `int a` `int b`

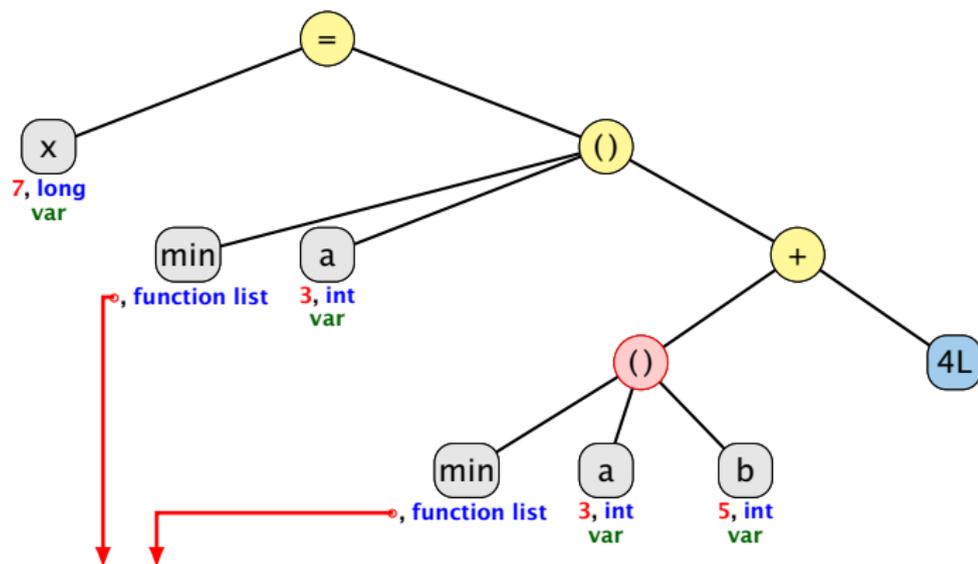
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

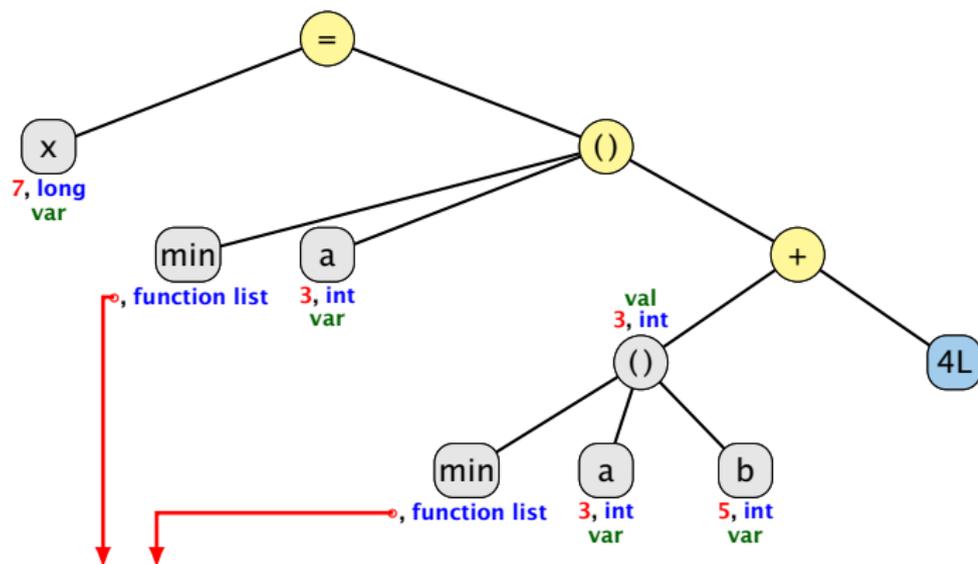
Beispiel: $x = \min(a, \min(a,b) + 4L)$



`int min(int,int)`
`float min(float,float)`
`double min(double,double)`

`long x` `7` `int a` `3` `int b` `5`

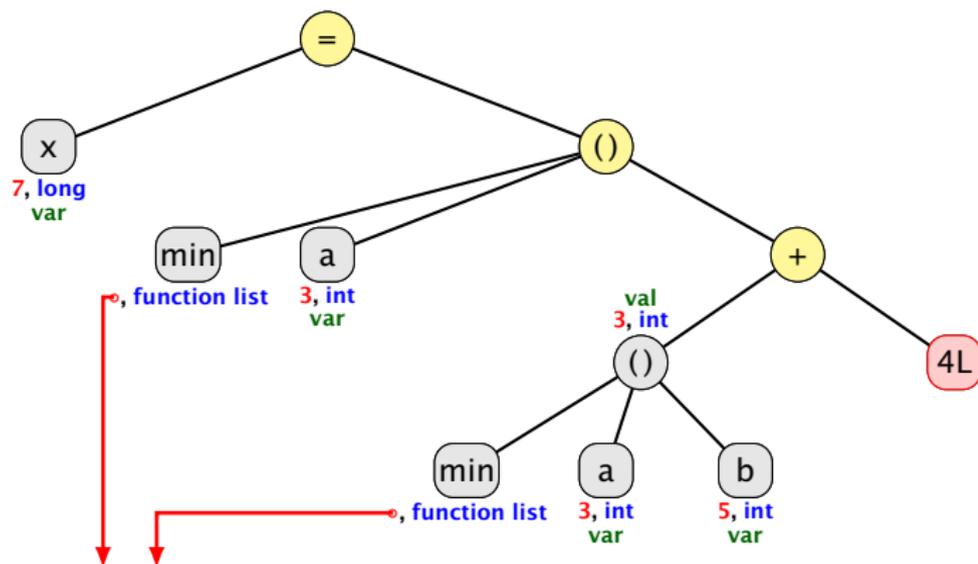
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

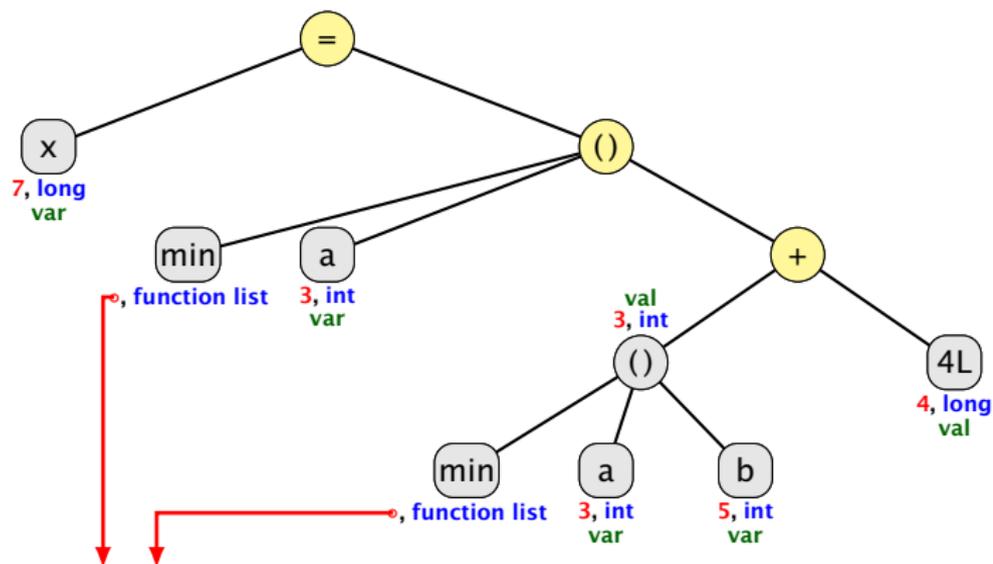
Beispiel: $x = \min(a, \min(a,b) + 4L)$



int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

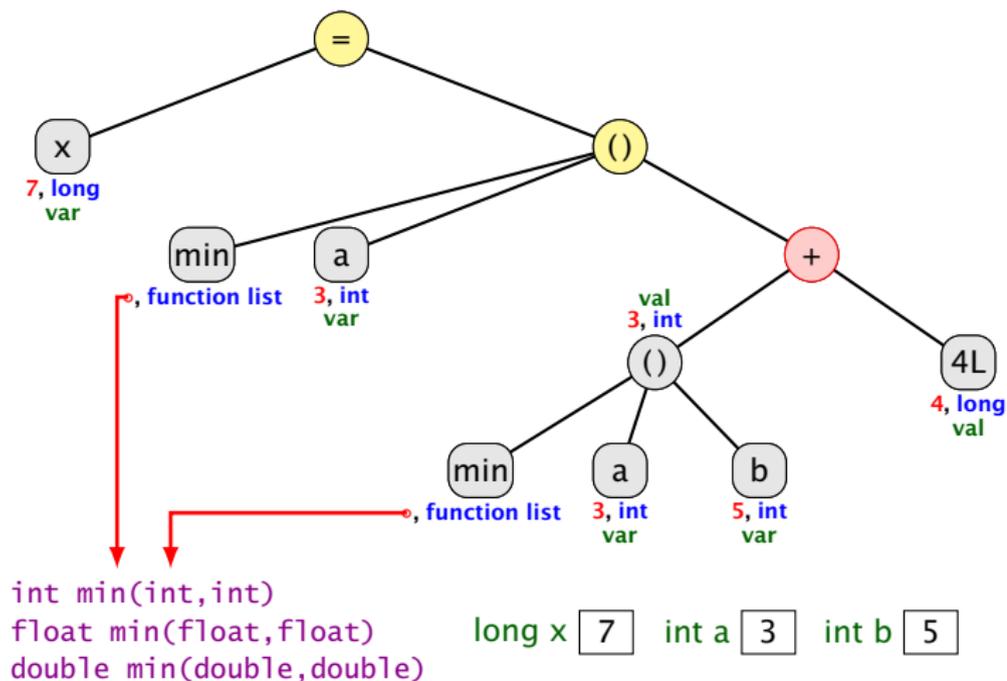
Beispiel: $x = \min(a, \min(a,b) + 4L)$



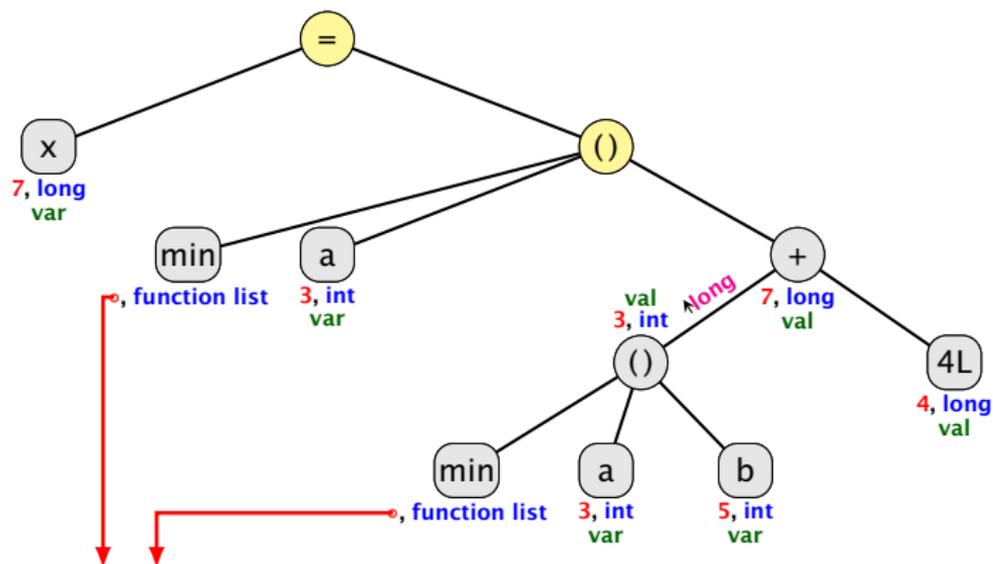
int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

Beispiel: $x = \min(a, \min(a,b) + 4L)$



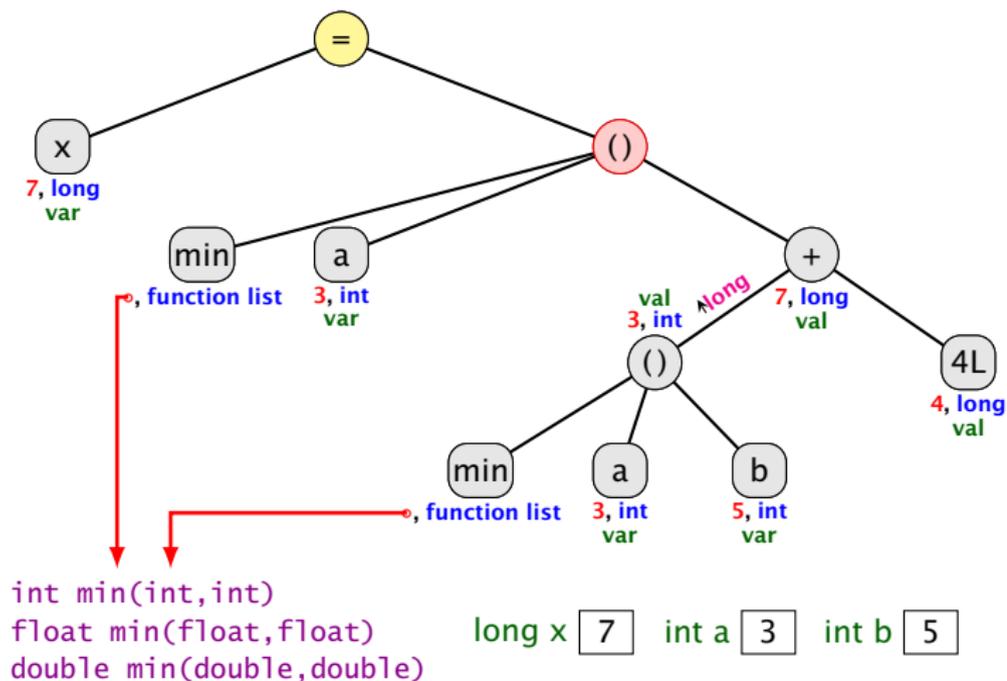
Beispiel: $x = \min(a, \min(a,b) + 4L)$



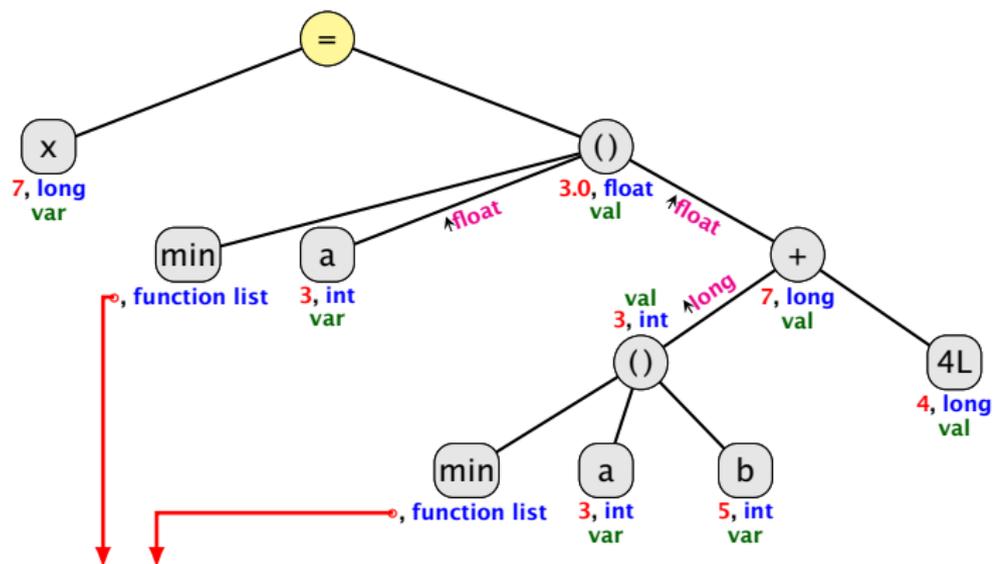
int min(int,int)
float min(float,float)
double min(double,double)

long x [7] int a [3] int b [5]

Beispiel: $x = \min(a, \min(a,b) + 4L)$



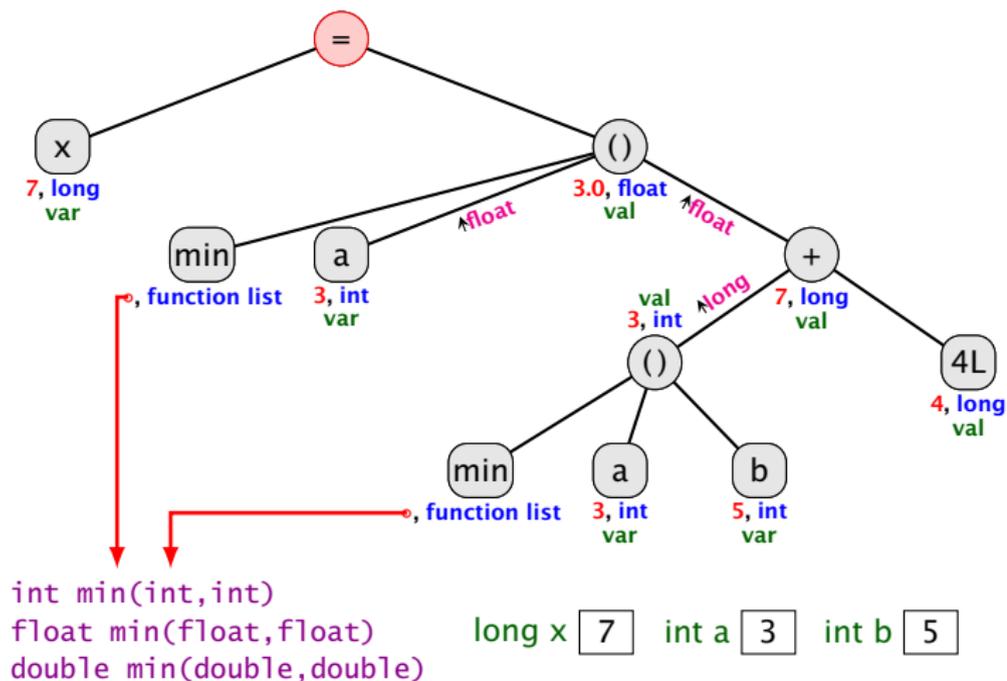
Beispiel: $x = \min(a, \min(a,b) + 4L)$



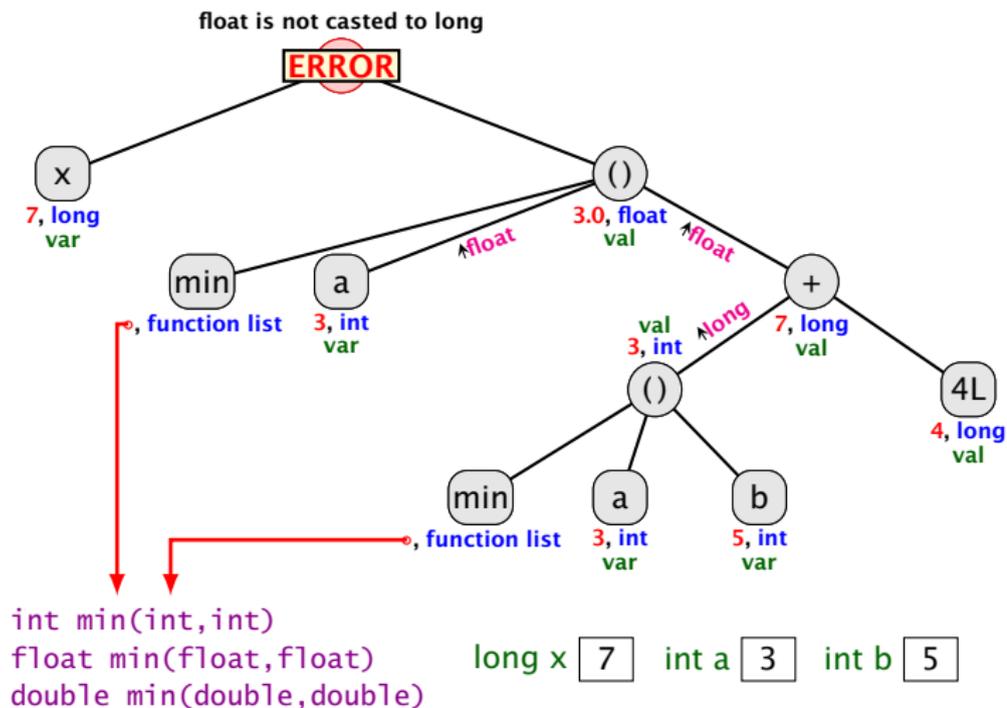
int min(int,int)
float min(float,float)
double min(double,double)

long x int a int b

Beispiel: $x = \min(a, \min(a,b) + 4L)$



Beispiel: $x = \min(a, \min(a,b) + 4L)$



Impliziter Typecast – Strings

Spezialfall

- ▶ Falls beim Operator `+` ein Typ vom Typ `String` ist, wird der andere auch in einen `String` umgewandelt.
⇒ Stringkonkatenation.
- ▶ Jeder Typ in `Java` besitzt eine Stringrepräsentation.

Funktioniert nicht bei selbstgeschriebenen Funktionen.

Beispiel: $s = a + b$

s = a + b

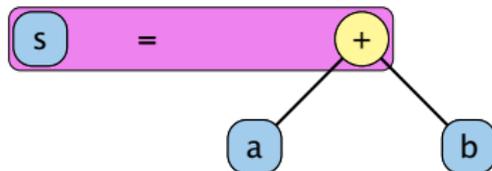
Beispiel: $s = a + b$



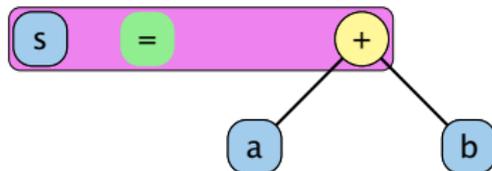
Beispiel: $s = a + b$



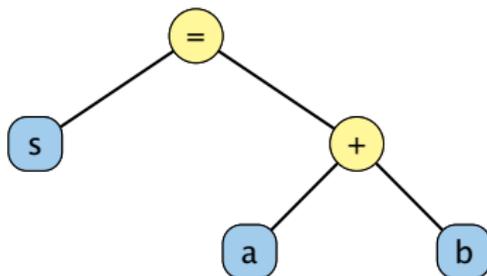
Beispiel: $s = a + b$



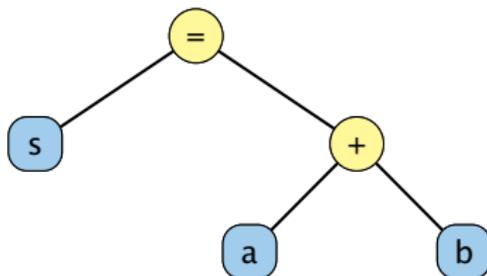
Beispiel: $s = a + b$



Beispiel: $s = a + b$

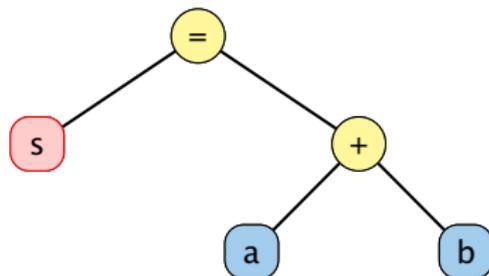


Beispiel: $s = a + b$



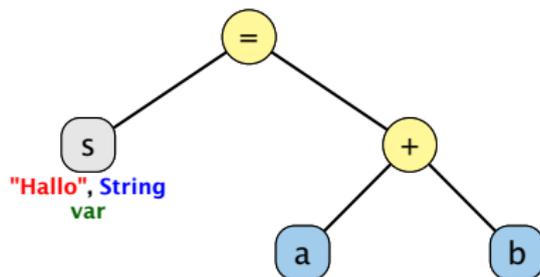
String s \rightarrow "Hallo" a 2 b 6

Beispiel: $s = a + b$



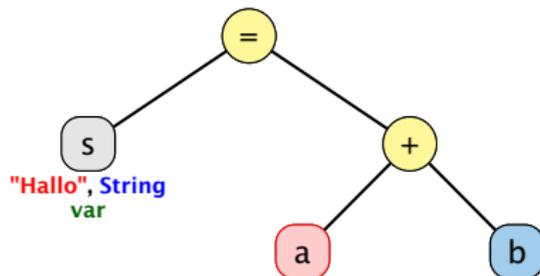
String s \rightarrow a b

Beispiel: $s = a + b$



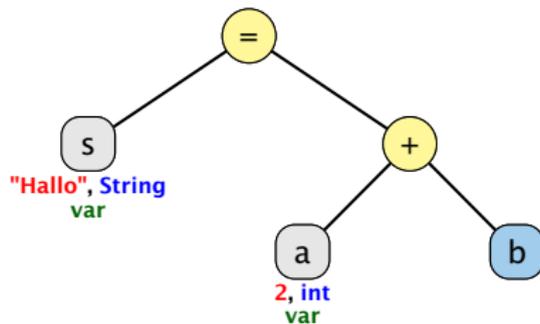
String s \rightarrow "Hallo" a b

Beispiel: $s = a + b$



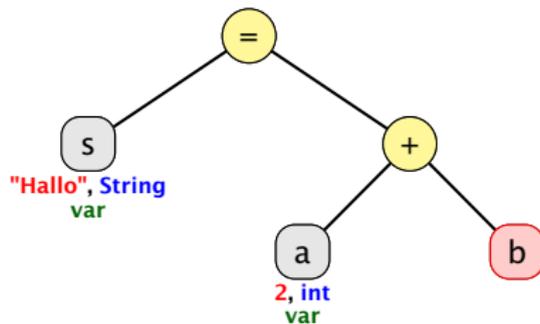
String s **→** a b

Beispiel: $s = a + b$



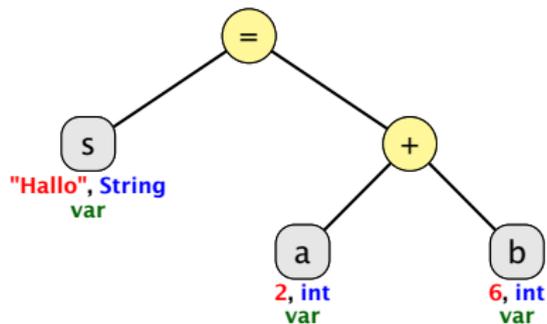
String s **→** "Hallo" a 2 b 6

Beispiel: $s = a + b$



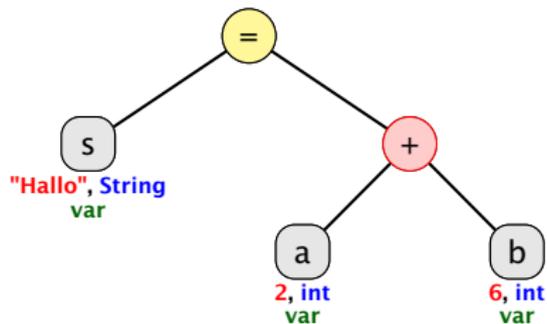
String s → "Hallo" a 2 b 6

Beispiel: $s = a + b$



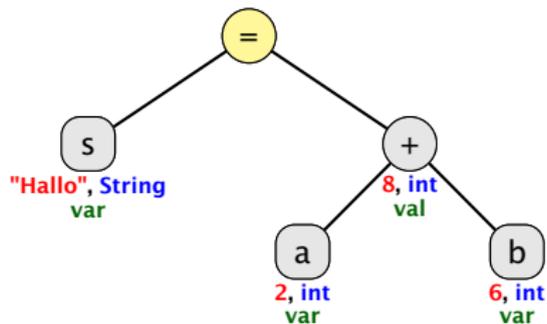
String s → "Hallo" a 2 b 6

Beispiel: $s = a + b$



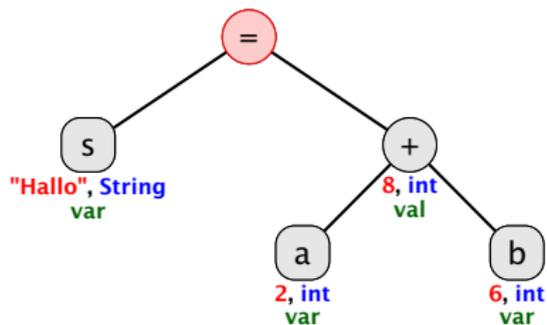
String s \rightarrow "Hallo" a 2 b 6

Beispiel: $s = a + b$



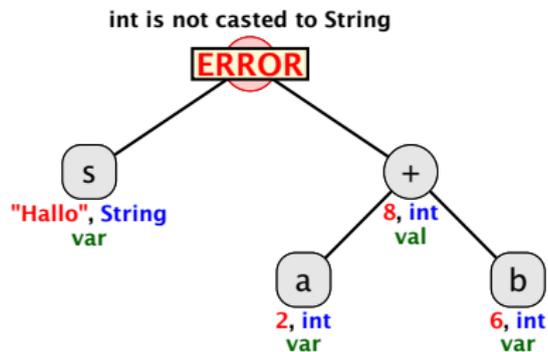
String s → "Hallo" a b

Beispiel: $s = a + b$



String s → "Hallo" a 2 b 6

Beispiel: $s = a + b$



String s → "Hallo" a b

Beispiel: $s = "" + a + b$

$s = "" + a + b$

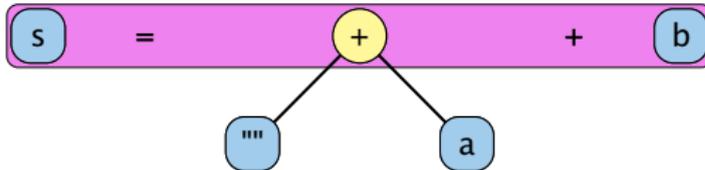
Beispiel: $s = "" + a + b$



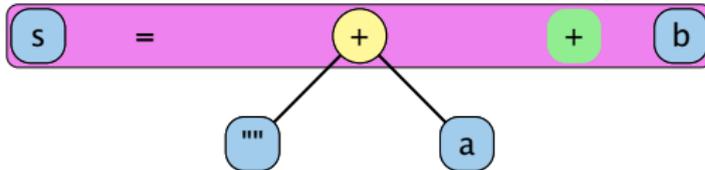
Beispiel: $s = "" + a + b$



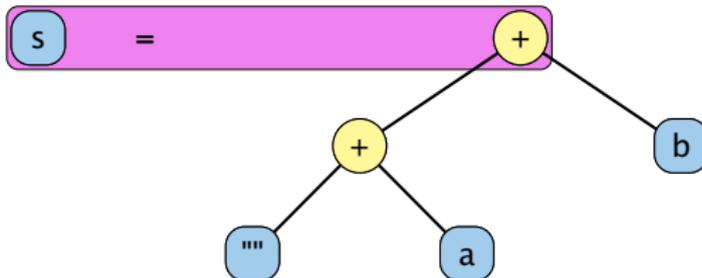
Beispiel: $s = "" + a + b$



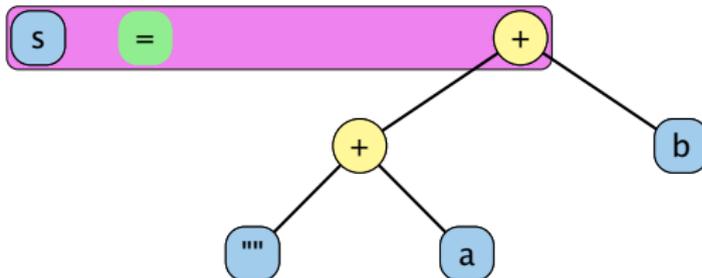
Beispiel: $s = "" + a + b$



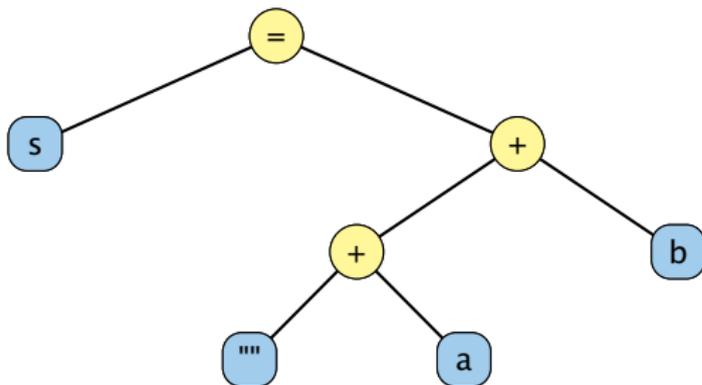
Beispiel: $s = "" + a + b$



Beispiel: $s = "" + a + b$

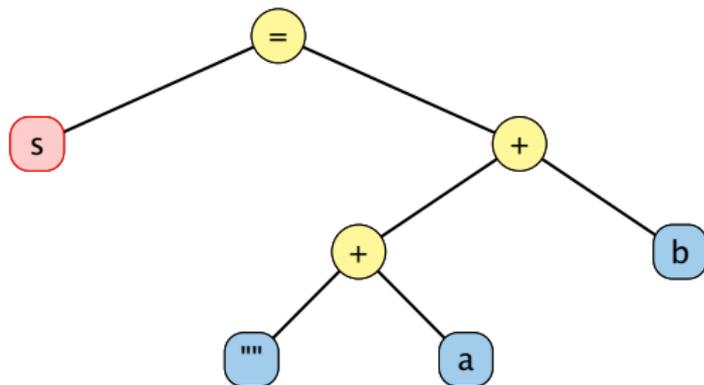


Beispiel: $s = "" + a + b$



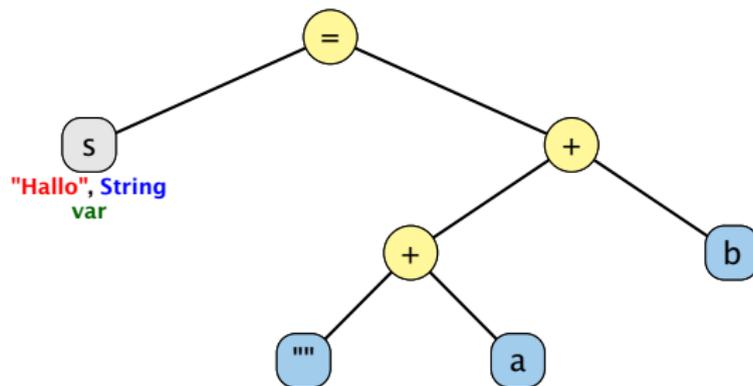
String s \rightarrow "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



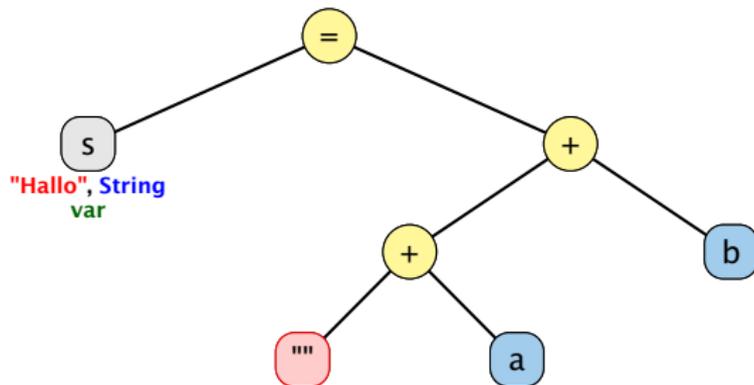
String s \rightarrow a b

Beispiel: $s = "" + a + b$



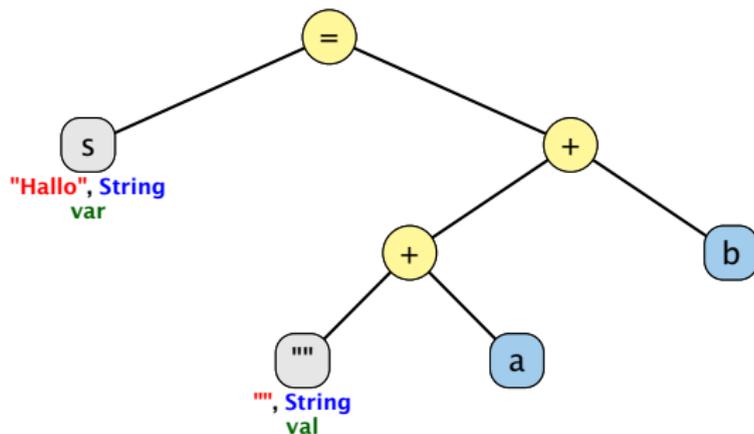
String `s` \rightarrow `"Hallo"` `a` `b`

Beispiel: $s = "" + a + b$



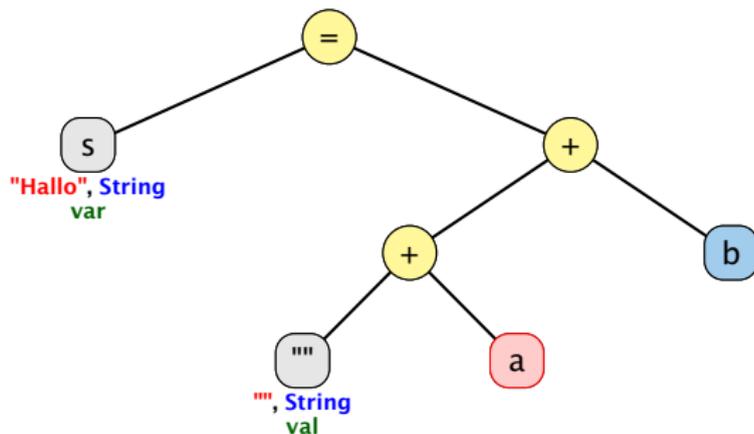
String s → "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



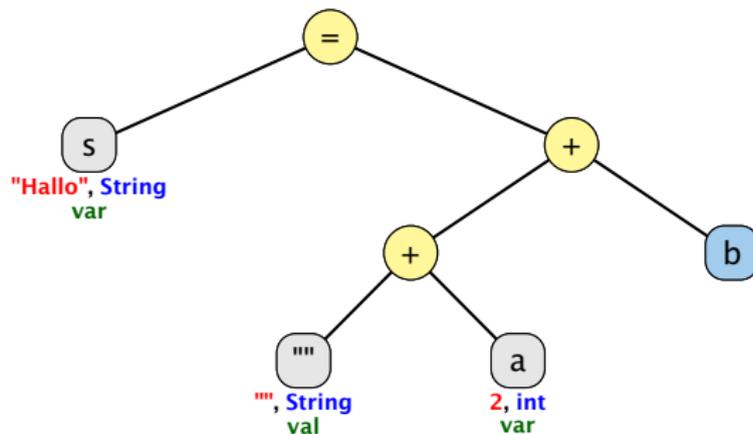
String s → "Hallo" a b

Beispiel: $s = "" + a + b$



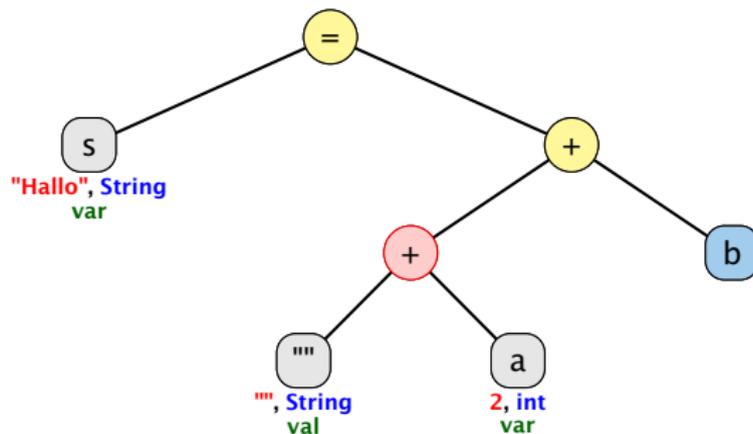
String s → a b

Beispiel: $s = "" + a + b$



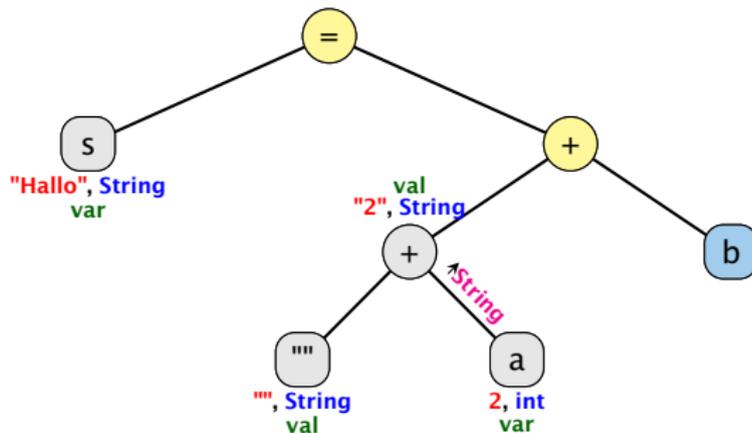
String s → "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



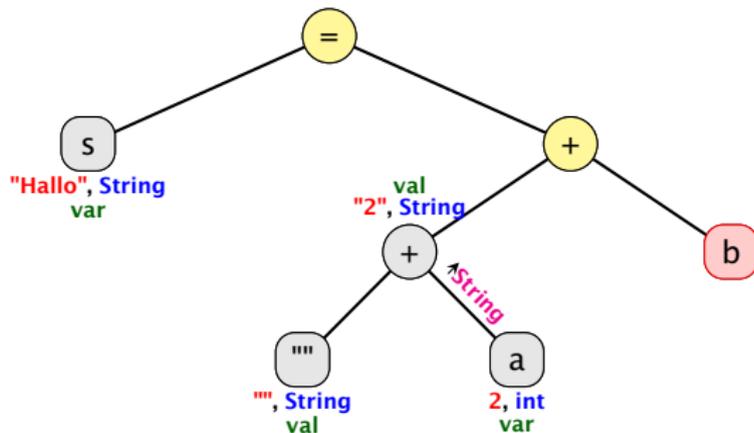
String s → a b

Beispiel: $s = "" + a + b$



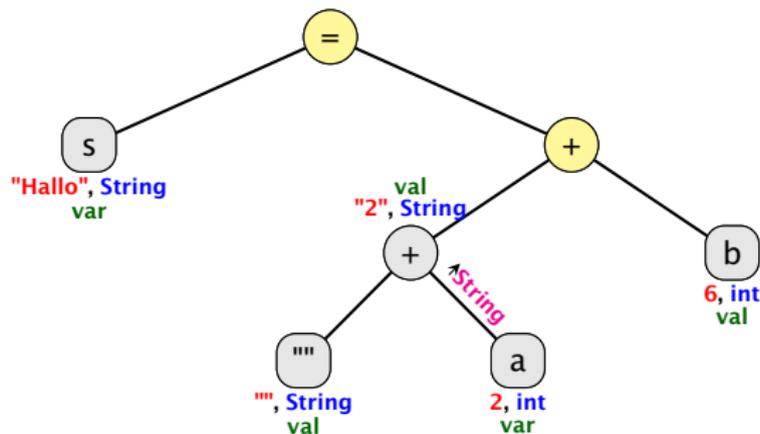
String s → a b

Beispiel: $s = "" + a + b$



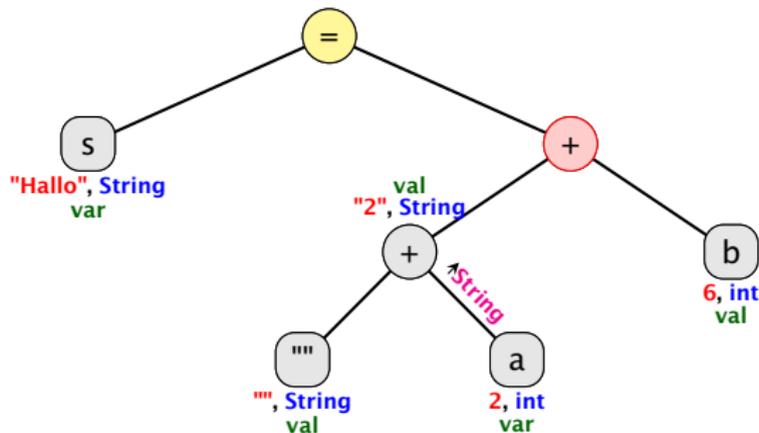
String s \rightarrow "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



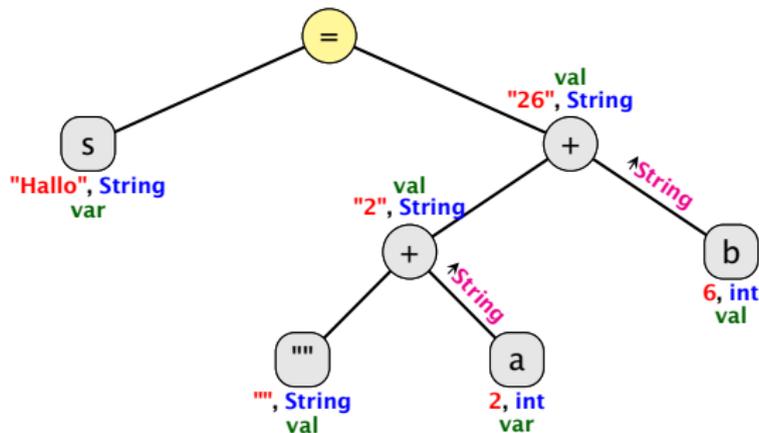
String s → a b

Beispiel: $s = "" + a + b$



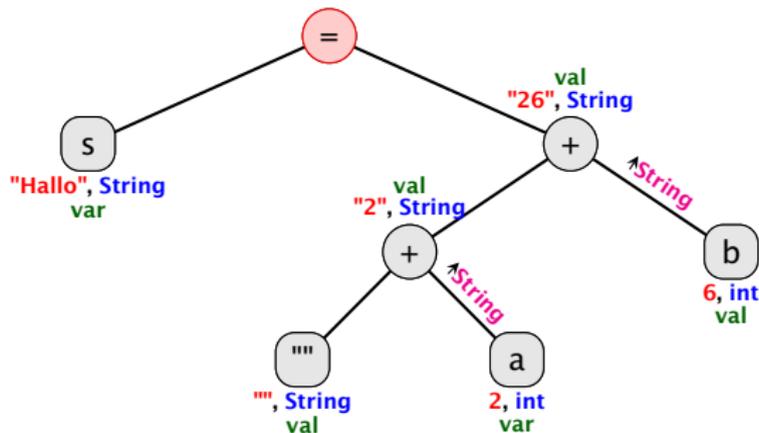
String s → a b

Beispiel: $s = "" + a + b$



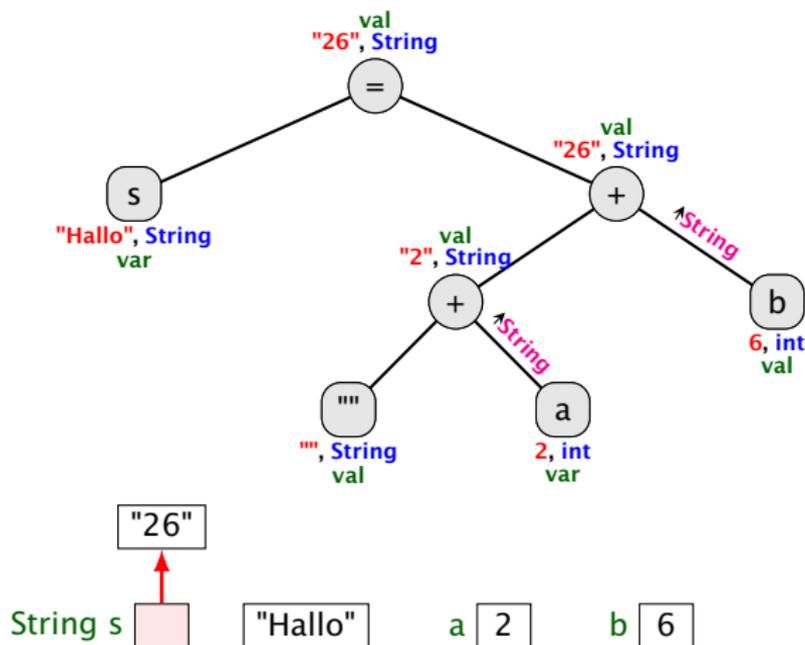
String s → "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



String s → "Hallo" a 2 b 6

Beispiel: $s = "" + a + b$



Beispiel: $s = s + 1$

$$s = s + 1$$

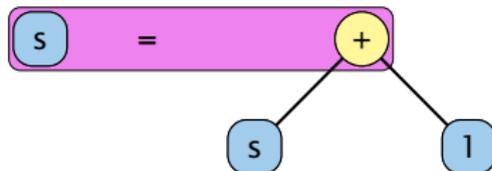
Beispiel: $s = s + 1$



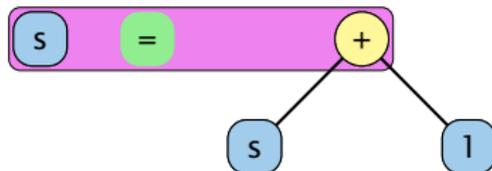
Beispiel: $s = s + 1$



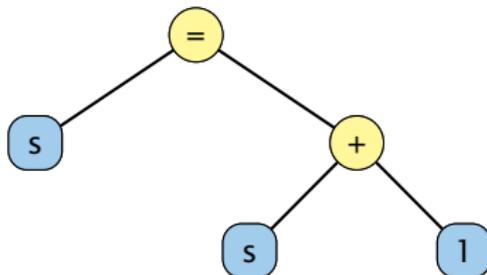
Beispiel: $s = s + 1$



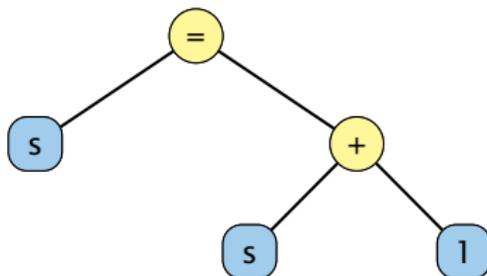
Beispiel: $s = s + 1$



Beispiel: $s = s + 1$

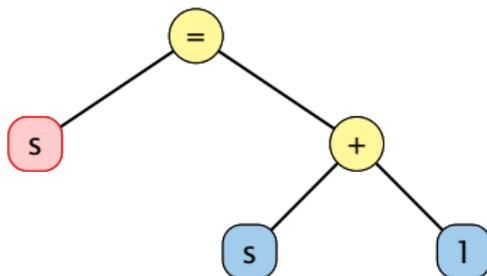


Beispiel: $s = s + 1$



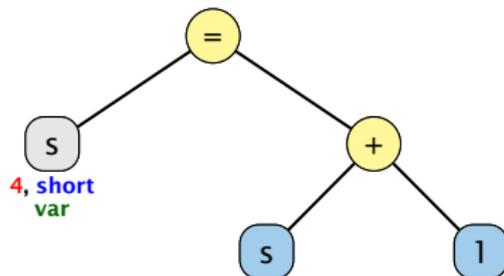
short s 4

Beispiel: $s = s + 1$



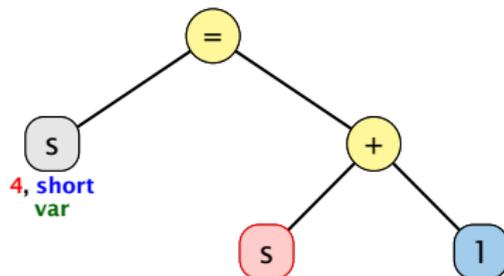
short s 4

Beispiel: $s = s + 1$



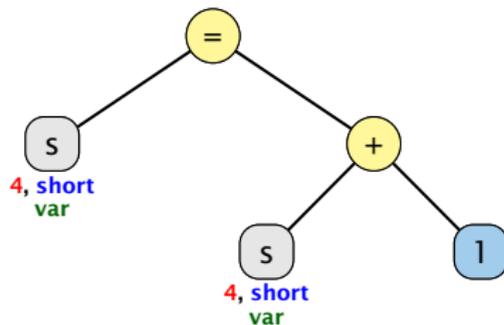
short s 4

Beispiel: $s = s + 1$



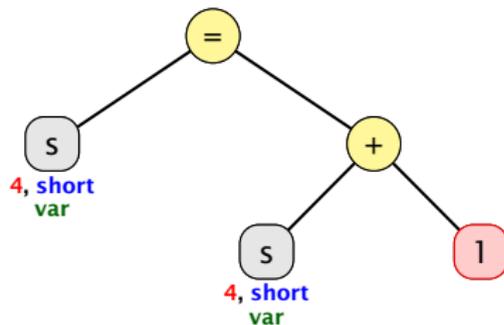
short s 4

Beispiel: $s = s + 1$



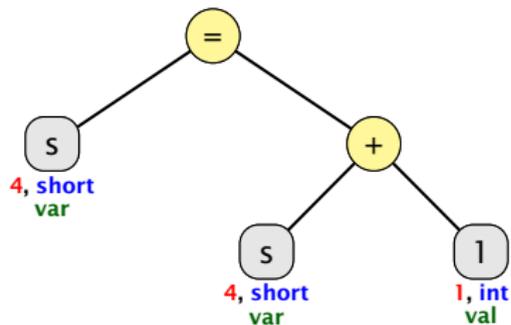
short s 4

Beispiel: $s = s + 1$



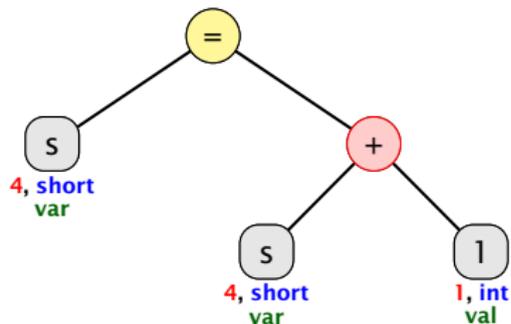
short s 4

Beispiel: $s = s + 1$



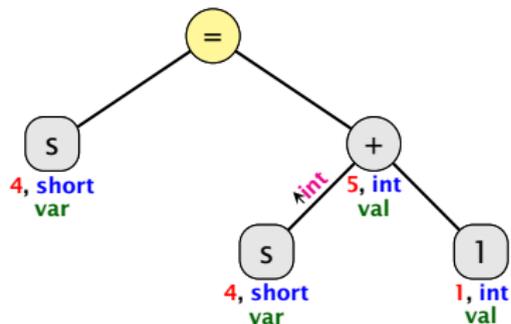
short s 4

Beispiel: $s = s + 1$



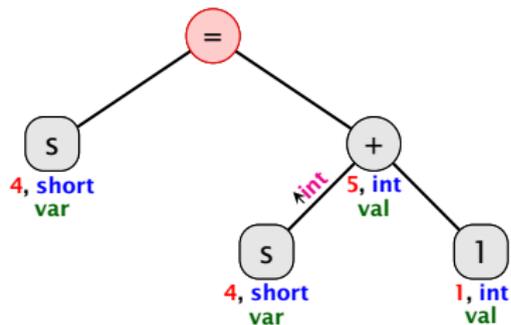
short s 4

Beispiel: $s = s + 1$



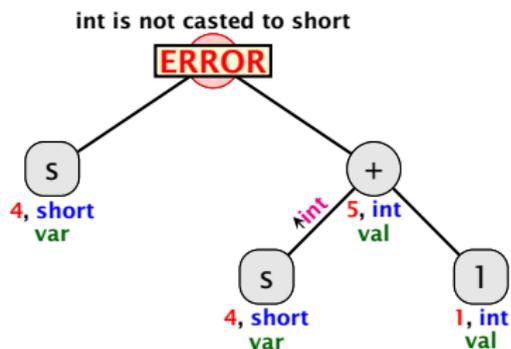
short s 4

Beispiel: $s = s + 1$



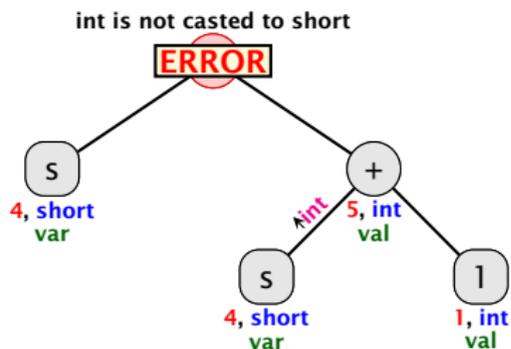
short s 4

Beispiel: $s = s + 1$



short s 4

Beispiel: $s = s + 1$



short s 4

Beispiel: $s = 7 + 1$

$$s = 7 + 1$$

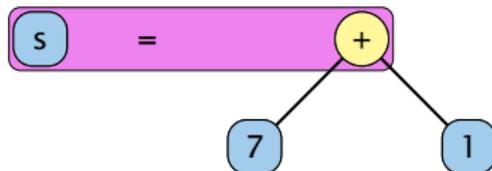
Beispiel: $s = 7 + 1$



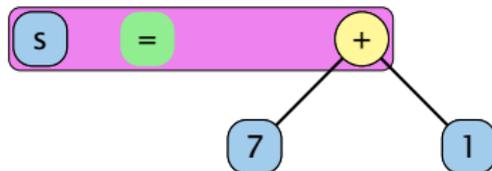
Beispiel: $s = 7 + 1$



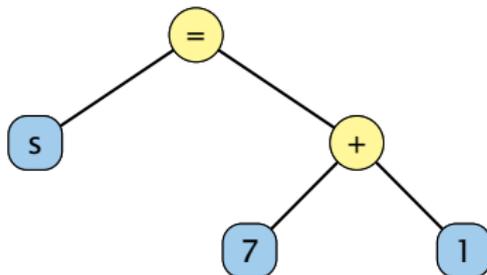
Beispiel: $s = 7 + 1$



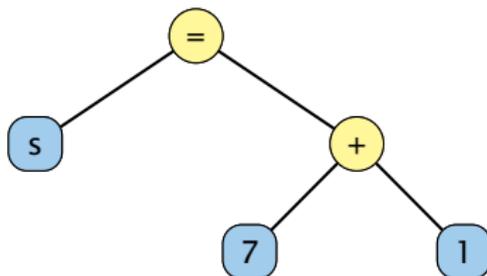
Beispiel: $s = 7 + 1$



Beispiel: $s = 7 + 1$

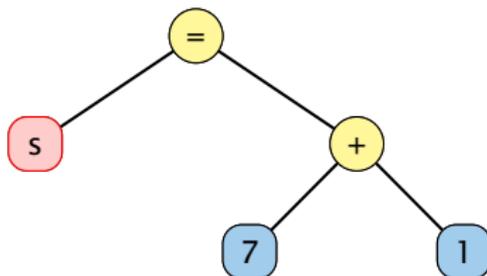


Beispiel: $s = 7 + 1$



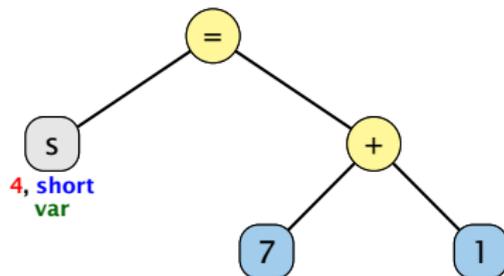
short s 4

Beispiel: $s = 7 + 1$



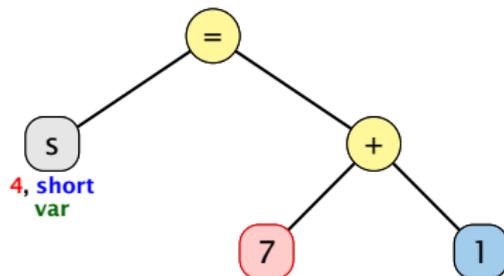
short s 4

Beispiel: $s = 7 + 1$



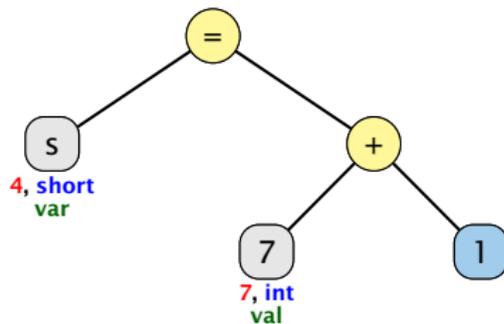
short s 4

Beispiel: $s = 7 + 1$



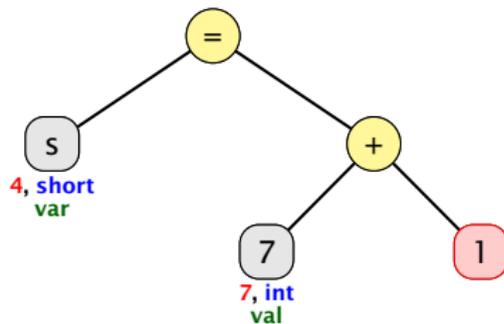
short s 4

Beispiel: $s = 7 + 1$



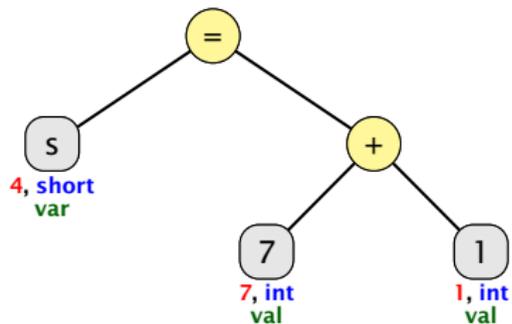
short s 4

Beispiel: $s = 7 + 1$



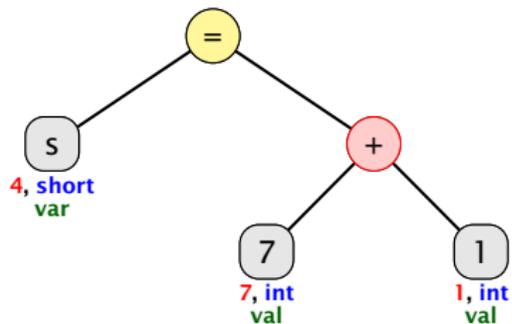
short s 4

Beispiel: $s = 7 + 1$



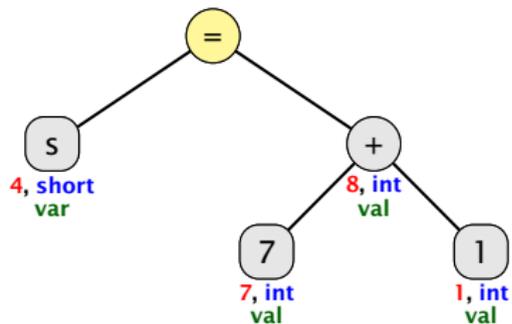
short s 4

Beispiel: $s = 7 + 1$



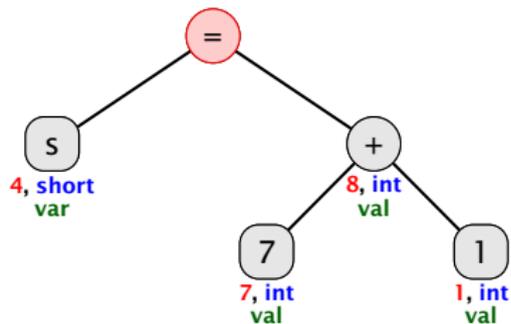
short s 4

Beispiel: $s = 7 + 1$



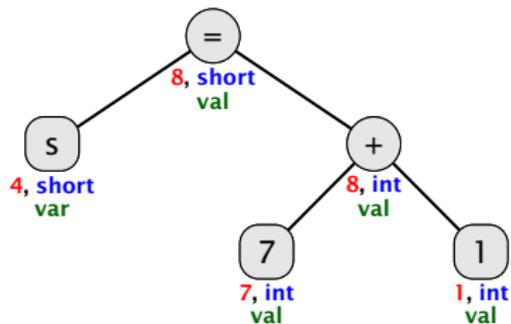
short s 4

Beispiel: $s = 7 + 1$



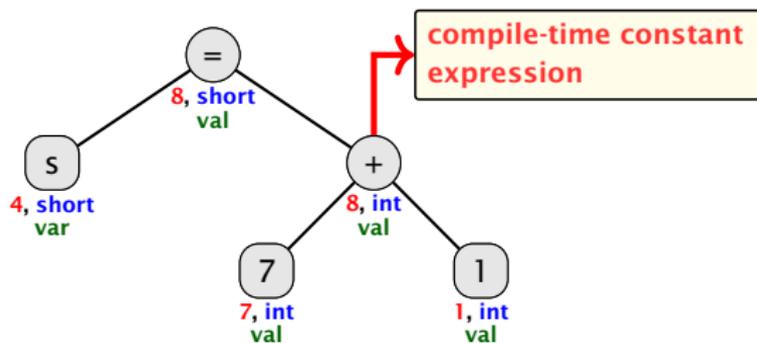
short s 4

Beispiel: $s = 7 + 1$



short s 8

Beispiel: $s = 7 + 1$



short s 8

Expliziter Typecast

<i>symbol</i>	<i>name</i>	<i>type</i>	<i>L/R</i>	<i>level</i>
(type)	typecast	Zahl, char	rechts	3

Beispiele mit Datenverlust

- ▶ `short s = (short) 23343445;`

Die obersten bits werden einfach weggeworfen...

- ▶ `double d = 1.5;`
`short s = (short) d;`

`s` hat danach den Wert **1**.

...ohne Datenverlust:

- ▶ `int x = 5;`
`short s = (short) x;`